Chapter 14
Real-world Service Interaction with Enterprise Systems in Dynamic Manufacturing Environments

S. Karnouskos, D. Savio, P. Spiess, D. Guinard, V. Trifa and O. Baecker

Abstract The factory of the future will be heavily based on internet and web technologies. A new generation of devices with embedded hardware and software will feature greatly improved storage, computing, and networking capabilities. This will lead to a system landscape of millions of networked devices that is heterogeneous with respect to functionality but features standard interfaces. This new breed of devices will not only be able to store and report information about themselves and their physical surroundings, but execute more computations and local logic. They will form collaborative peer-to-peer networks and also connect to central systems. By eliminating media breaks, e.g. by replacing manual data entry with a direct connection to devices, this "internet of things" will feature end-to-end connectivity, making the models of the real world, as they exist in business systems, follow reality more precisely and with shorter delay. This will change the way we design, deploy and use services at all layers of the system, be it the device, line, plant, or company level or even between collaborating organizations. This chapter describes an architecture for effective integration of the services from the internet of things with enterprise services. We describe the case of centrally managing a population of devices that are located at different sites, including dynamic discovery of devices and the services they offer, near real-time cross-site interaction, interaction with business processes and distributed system management.

Keywords Service interaction, enterprise integration, dynamic manufacturing

S. Karnouskos (✉), D. Savio, P. Spiess, D. Guinard, V. Trifa and O. Baecker
SAP Research, SAP AG, Vincenz-Priessnitz-Strasse 1, D-76131 Karlsruhe, Germany
e-mail: stamatis.karnouskos@sap.com

14.1 Motivation

The last decade has witnessed a deep paradigm shift on the shop-floor where information and communication technologies are being used extensively. As high-performance micro-controllers are being embedded in devices used in manufacturing and process automation, services hosted on them will enable new applications that could significantly increase the efficiency and efficacy of current shop-floor systems.

As we are moving towards the "internet of things" as depicted by Fleisch and Mattern (2005), millions of interconnected devices will provide and consume information available on the network and cooperate. Service-oriented architecture (SOA) seems to be a promising solution to realize the necessary universal interoperability. SAP predicts that the world market for technologies, products, and applications that are related to the "internet of things" will increase significantly from €1.35 billion to more than €7.76 billion in 2012, with average annual growth rates of almost 50% (SAP, 2008).

In what we call "real-world SOA", each device:

- offers its functionality in a service-oriented way;
- is able to discover other devices and their hosted services dynamically at runtime;
- can invoke actions of the discovered services dynamically; and
- is able to publish and subscribe to typed, asynchronous events.

These distributed devices can be considered as a set of intelligent, proactive, fault-tolerant and reusable units (Colombo and Karnouskos, 2009) that can co-operate to form a dynamic infrastructure able to provide better insights of the current status of, e.g., a production line to the other higher levels in the factory information technology (IT) systems, such as manufacturing execution systems (MES). They also can dynamically react to business changes that can influence the production plan on the shop-floor.

As demonstrated in previous work (de Souza *et al.*, 2008; Jammes and Smit, 2005; Karnouskos *et al.*, 2007; Priyantha *et al.*, 2008), future shop-floor infrastructures can significantly benefit from service-oriented approaches, both in vertical (cross-level) and horizontal communication, as pursued within the SOCRADES project (www.socrades.eu). In these infrastructures, new, rich services can be created by orchestrating and combining services from different system levels, i.e. services provided by enterprise systems, by middleware systems on the network, and by devices themselves. The composed services with complex behaviour can be created at any layer (even at device layer). In parallel, dynamic discovery and peer-to-peer (P2P) communication allows for dynamically discovery of functionality of each device. The trend is to clearly move away from proprietary connections between monolithic hardware and software systems towards more autonomous systems that interact in a more standardized, cooperative and open way.

The convergence of applications and products towards the SOA paradigm improves shop-floor integration and transparency, thereby increasing reactivity and performance of the workflows and business processes commonly found in manufacturing and logistics. Events become available to any entity of the system as they happen, and business-level applications can exploit such timely information for purposes such as diagnostics, performance indications, or traceability. While these vertical collaborations are beneficial for business application software, new challenges arise: direct communication with devices can be error prone or unreliable, which must be considered when critical decisions, such as branches in a workflow, depend on it.

Business processes in a company are defined by the best practices of the respective industry and its goals. However, in reality, production processes are monolithic and system output is expected to be ideal. A production process instance (residing, e.g., in an MES system) usually has a series of vertical integrations with shop floor systems until it reaches the end of its lifetime. As a consequence, challenges arise when trying to make the processes adaptive or trying to extend it. Introducing process parameters to adapt to the dynamic nature of the shop-floor is tedious, especially for companies that span multiple production locations and heterogeneous IT systems.

The embedded device landscape is changing drastically as technology rapidly advances. Shop floors become populated with highly sophisticated networked embedded devices that have faster central processing units with lower energy consumption, yet are more compact. They can also do more than controlling local loops and can provide tools for real-time analysis (for example the CX1020 series programmable logic controller (PLC) of Beckhoff).

As devices can natively offer web services, they provide an interoperability layer that leads to easier coupling with other components despite of the high heterogeneity behind the web service facades. Device profile for web services (DPWS) Chan *et al.*, (2005), OPC-UA (Mahnke *et al.*, 2009) and representational state transfer (REST) are three of the emerging technologies for realizing web service-enabled devices. Thanks to the nature of web services, compositions of services can be easily created to match the desired scenario, very much like in state-of-the-art web mashups. Integration of devices at the functional level allows us to focus on orchestrating services based on their role in a process, and not the low-level interface of the device as such.

Service-based integration of shop-floor devices with enterprise systems brings many benefits in terms of business automation, response time, and data quality. Although these benefits make this integration highly desirable in a competitive economy, the unsupervised integration of devices with enterprise systems can also cause economic losses. These losses include: production halts, production time increase, reputation loss due to delays and even product recalls. When unexpected situations occur on the shop-floor, a rapid and dynamic adaptation of the business process is required in order to mitigate the effects that such an event can cause.

Hence, a beneficial integration of shop-floor devices with enterprise systems should provide characteristics that enable business processes to dynamically adapt to changes in the state of the device layer. With the current improvement of shop-floor devices and the adoption of SOA on all layers of the system, it is possible to create systems that are self-healing, self-monitoring, and self-optimizing.

Our aim is to depict how we can move towards highly dynamic manufacturing enterprises. Although simulations have been used before either at the shop-floor or at business process level, they have been used in an isolated way. In our work, we try to integrate the shop-floor and the business system and dynamically assess the state of the resulting, holistic system. With the help of simulations and monitoring, enterprises can adapt to situations and predict possible problems on the shop floor. SOA-based middleware, such as one we proposed in Karnouskos *et al.* (2007) and demonstrated in de Souza *et al.* (2008), has lead to preliminary promising results in this area. We propose to extend our SOA-based system to accommodate simulation and analytics as well as decision-making and process-mapping strategies. This helps enterprise systems to dynamically adapt to changes in the shop-floor, to reduce the gap between the real world and its digital representation, and also to optimize business processes.

14.2 Real-world Awareness

Significant effort has been invested into the integration of physical computing devices with standard enterprise software, such as enterprise resource planning (ERP) systems. Planning a production order or creating a bill of materials in the ERP application is neither effective nor optimized, unless the shop-floor is transparent. As an example, the manufacturing industry foresees enterprise applications to consider real-time events on the shop-floor to plan production, enhance customer relationship management, and have a healthy updated supply chain. This shop-floor intelligence obtained in real time allows business to adapt to the market demand and forecast shop-floor breakdowns in a timely fashion. Additionally as SOA approaches start to prevail (Kennedy *et al.*, 2008), the introduction cycle of new applications could be significantly shorter. This could enable exchange of real-time information across enterprises and trusted business partners, which will have an effect on the respective business decisions.

14.2.1 Device Integration Protocols

Shop-floor integration protocols based on web services as for example OPC UA (Mahnke *et al.*, 2009) are emerging in the automation domain. Until now, advanced features like dynamic device discovery, eventing and notification mecha-

nisms are only on conceptual level in OPC UA specifications. Hence, OPC UA-based clients have to be installed on all systems that would need to consume shop-floor data from a device hosting the services. Furthermore the web service part of OPC UA is optional and performance evaluation or real-world experiences are not available to our knowledge.

Other shop-floor integration standards in the semiconductor industry are available, for example the SEMI (www.semi.org). Equipment Communications Standard/Generic Equipment Model (SECS/GEM) communication protocols connect a host computer and semiconductor manufacturing equipment. Photovoltaic Equipment Communication Interfaces is based on the SECS/GEM and targeted towards the photovoltaic industry. Clients have to be implemented that understand the protocols and communicate using interfaces defined in the standards. The biggest disadvantage is that only one client can interact with a server in a session. Multiple sessions are not possible. As such any approach to enable the service-oriented paradigm over this protocol would be very difficult to achieve without significant extensions.

Another standard dealing with ubiquitous device integration is DPWS (Chan *et al.*, 2005), which is a collection of web service standards. Initially, DPWS was conceived as a successor of universal plug-and-play (UpnP) for home automation scenarios, but recent work has shown its applicability to the automation world (Jammes and Smit, 2005). DPWS advances previous dynamic discovery concepts such as Jini (www.jini.org) and UPnP (www.upnp.org) to integrate devices into the networking world and make their functionality available in an interoperable way. DPWS is an effort to bring web services to embedded devices taking into consideration their constrained resources. Several implementations exist in Java and C (e.g. www.ws4d.org, www.soa4d.org), while Microsoft has also included a DPWS implementation (WSDAPI) by default in Windows Vista® and Windows Embedded CE operating systems. As depicted in Figure 14.1, any DPWS-empowered device can be dynamically discovered with existing Windows Vista® installations and its basic metadata can be read.

An alternative integration approach is REST (Fielding, 2000), which is the architectural principle that lies at the heart of the Web and shares a similar goal with integration techniques such as (DP)WS-* web services, that is increasing interoperability for a looser coupling between the parts of distributed applications. However, the goal of REST is to achieve this in a more lightweight and simpler manner; therefore it focuses on resources, and not functions as is the case with WS -* web services. In particular, REST uses the Web as an application platform and fully leverages all the features inherent to HTTP such as authentication, authorization, encryption, compression, and caching. This way, REST brings services “into the browser”, i.e., resources can be linked and bookmarked and the results are visible with any web browser. There is no need to generate complex source code out of WSDL files to be able to interact with the service.

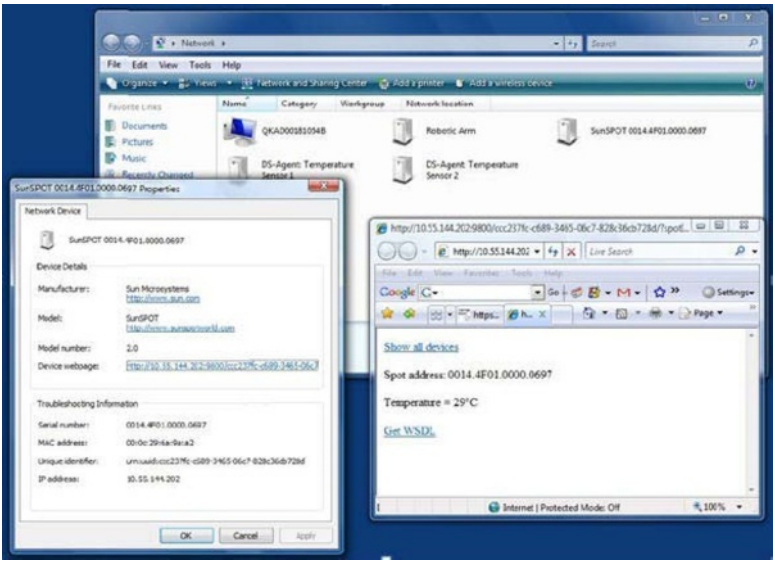

Fig. 14.1 DPWS-based dynamic discovery of devices built in Windows Vista®

14.2.2 Device-to-Business Coupling

In the area of enterprise application integration, a few projects have explored the use of "mashups", also known as user-generated composite applications, to enable more flexible software composition within (and outside) the enterprise (Hoyer *et al.*, 2008; Liu *et al.*, 2007). However, they mainly focus on mashing up on-line services and do not address the issues and requirements that come with a physical world integration (e.g., as discussed in de Souza *et al.*, 2008; Marin-Perianu *et al.*, 2007). To ensure interoperability across all systems, recent work has focused on applying the concept of SOA; in particular web services standards (SOAP, WSDL, etc.) directly on devices (Jammes and Smit, 2005; Karnouskos *et al.*, 2007; Priyantha *et al.*, 2008). Implementing WS-* standards on devices presents several advantages in terms of end-to-end integration and programmability by reducing the need for gateways and mediators between the components. This enables the direct orchestration of services running on devices, with high-level enterprise services. As an example, if sensors physically attached to shipments could offer their functionality via web services; they could be easily discovered and integrated in a process that updates the status and location of the shipment directly in the involved ERP systems.

As an alternative to WS-* standards, more "web-oriented" (sometimes called "web of things" as opposed to the "internet of things" (Guinard and Trifa, 2009)) approaches are emerging, i.e. approaches (re)using standards on the web and ap-

plying them to real-world devices for them to provide services directly on the web. As an example, web feeds have been used to access data provided by sensor nodes (Dickerson *et al.*, 2008). In particular, they describe an extension to RSS better suited to accommodate high-rate data streams with a web-oriented querying interface to retrieve sensor data. A direct consequence of the stream abstraction is that sensors are considered solely as data publishers, not as service providers. Further examples (Drytkiewicz *et al.*, 2004; Guinard *et al.*, 2009; Luckenbach *et al.*, 2005; Wilde, 2007) propose and evaluate a RESTful (Fielding, 2000) architecture for sensor networks.

14.2.3 Integrating Heterogeneous Devices

In de Souza *et al.* (2008) and Karnouskos *et al.* (2007) we proposed an extensible integration architecture based on web services and capable also of supporting legacy products. There are basically three directions we follow as shown on Figure 14.2:

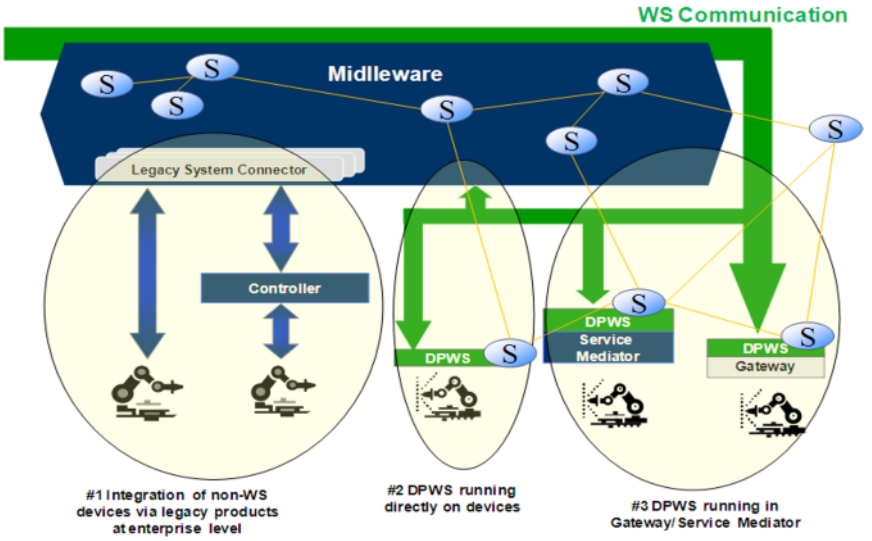

Fig. 14.2 Integration approaches to couple legacy and emerging device infrastructure to enterprise systems

- Integration via legacy products: several tools are available to the market today, so we rely to them for providing the connectivity (as it is done today).
- Integration via gateways and service mediators: legacy or resource-scarce devices have their functionality wrapped as a web service at a higher layer, e.g., a control point (gateway approach) or their functionality is aggregated/composed

and a new service depending on (possibly many devices) is created (mediator approach).
- Web service enabled devices: these have the computational power and communication capabilities to natively run a stack that provides their functionality as a set of web services. These devices can be directly discovered and can interact with each other.

14.3 Enterprise Integration

The SOCRADES integration architecture (SIA as depicted in Figure 14.3) enables enterprise-level applications to interact with and consume data from a wide range of networked devices using a high-level, abstract interface that features web service standards. Those standards already constitute the standard communication method used by the components of enterprise-level applications. Web services are the technology canonically used to implement business processes, which are frequently modelled as orchestrations of available web services. This allows the connected, networked devices to directly participate in business processes while neither requiring the process modeller nor the process execution engine to know about the details of the underlying hardware.

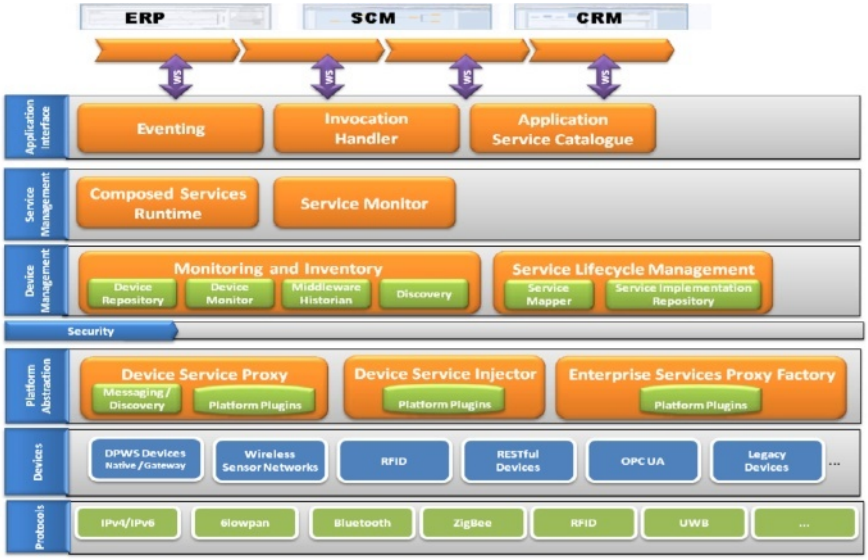

Fig. 14.3 The SOCRADES integration architecture

The requirements we want to cover as well as the functionality to be realized is analysed in de Souza *et al.* (2008) and Karnouskos *et al.* (2007). The architecture

implemented hides the heterogeneity of hardware, software, data formats, and communication protocols that is present in today's embedded systems. The following layers can be distinguished: application interface, service management, device management, security, platform abstraction, and devices.

Application interface: this part of the integration architecture features a messaging (or eventing) system, allowing an application to consume any events whenever it is ready to and not when a low-level service happens to send them. A so-called invoker allows buffering invocations to devices that are only intermittently connected. Finally, a service catalogue enables human users and applications to find service descriptions and pointers to running service instances. Both atomic services hosted by the devices and higher-level composed services are listed here.

Service management: all functionality offered by networked devices is abstracted by services. Either devices offer services directly or their functionality is wrapped by a service representation. On this layer of the integration architecture and on all layers above, the notion of devices is abstracted from and the only visible assets are services. An important insight into the service landscape is to have a repository of all currently connected service instances. This is provided by the service monitor.

This layer also provides a run-time for the execution of composed services. We support the composition of business processes *primarily* by offering an execution service for underspecified BPEL processes, meaning that service compositions can be modelled as business processes where the involved partners do not need to be explicitly specified at design time.

Device management: all devices are dynamically discovered, monitored and their status is available to the enterprise services. Furthermore, it is possible to remotely deploy new services during run-time, in order to satisfy application needs.

Security: both devices and back-end services may only be accessed by clients that have a certain role and provide correct credentials that authenticate themselves. This layer implements the correct handling of security towards the devices and the enterprise-level applications.

Platform abstraction: as stated before, devices either offer services directly or their functionality is wrapped into a service representation. This wrapping is actually carried out on the platform abstraction layer. In the best case, a device offers discoverable web services on an internet protocol (IP) network. In this case, no wrapping is needed because services are available already. If the device type, however, does not have the notion of a service (it might use a message-based or data-centric communication mechanism), the abstraction into services that offer operations and emit events can be a complex task. In addition to service-enabling the communication with devices, this layer also provides a unified view on remotely installing or updating the software that runs on devices and enables the devices to communicate natively, i.e. in their own protocol with back-end devices.

Devices: heterogeneous devices are expected to connect to the architecture. These include industrial devices, home devices, or IT systems such as mobile phones, personal digital assistants, production machines, robots, building automa-

tion systems, cars, sensors and actuators, radio-frequency identification (RFID) readers, barcode scanners, or power meters. We used several of the listed types of devices during prototype implementations as shown in the demonstration section.

A single lightweight component called the local discovery unit (LDU) is implemented. It can be downloaded or deployed on any Java-supported device. This component dynamically discovers devices with respect to the protocols it supports and connects them to SIA. We can therefore realize very dynamic scenarios, where an LDU for IP is downloaded and run in a mobile phone, and via its WiFi interface dynamically finds and uses the functionality of all DPWS-based devices on the local network. Several LDUs can be deployed on a network allowing for load balancing as well as wide protocol coverage. Finally, using diverse LDUs, the SIA is able to realize cross-network and even cross-enterprise dynamic device discovery and act as mediator of their services. Apart from the LDU, the rest of the components have been implemented based on open web service standards in Java EE 5 on the SAP NetWeaver CE platform, using EJB 3.0 and JPA.

Protocols: We expect that heterogeneity in communication protocols will exist also in the future as it serves different domains and respectively devices.

14.4 Integrating Manufacturing Equipment with the SOCRADES Integration Architecture

Traditionally, shop-floor data has been integrated with enterprise applications like ERP using proprietary, custom made, individually developed technologies, e.g. in the form of file transfers and proprietary middleware layers.

Standard connectors from the OPC Foundation for MES and Distributed Control Systems lowered the complexity of integration by providing a unified data format, but each connection must still be statically tailored for a particular group of devices of a certain vendor because the standard does not define the meaning of data points. If the same resource has to be used from different system layers or in a P2P way by other systems on the shop floor, appropriate clients have to be developed for each interface. Additionally, the old OPC specifications were tightly coupled with the DCOM technology, making adopters dependent on the Windows family of operating systems from Microsoft. In 2006, the OPC Foundation started specifying a new interface standard, OPC UA, as a successor of the native OPC standards based on COM/DCOM and mitigating some of the mentioned pain points. However, the idea of using plain web service on devices, creating SOA on the shop floor, is not implemented well in the new version of the interface standard. The web services are specified as an optional method to access OPC UA interfaces. A binary transport mechanism is mandatory for OPC UA devices to communicate. It specifies a very complex metamodel for data and service modeling that makes the implementation of servers and clients quite challenging. The added value over plain web services is questionable. Well-documented web ser-

vice interfaces could give the same value at a fraction of the complexity and still allow for cross-platform interoperability, flexibility, and dynamic processes.

In the business software domain, web services were originally designed to allow cooperation between applications within and across companies, possibly from different vendors, implemented using different programming languages, and running on different operating systems. They bring more flexibility and interoperability to cross-enterprise transactions that required constant change.

SAP ERP uses iDOCs to pass data between systems. iDOCs are (potentially large) extensible markup language (XML) documents, which contain data relevant for business transactions. In more recent versions of the application, using an enterprise SOA approach, these transactions are exposed as web services. A central universal description discovery and integration registry keeps track of services that can be consumed across the company. This enables ERP transactions to be more atomic, stand alone, and stateless. SAP ERP systems also provide services, which are harmonized with enterprise models based on process components, business objects and global data types. They follow well-defined common rules of business content. Web service-enabled devices on the shop floor would host services allowing them to directly monitor and influence physical processes. But the fragile nature of a device and its environment can influence the services hosted on the device. They lead to differences in reliability and availability (compared to regular enterprise services), which have to be considered when integrating devices on the shop floor with ERP platforms:

- Devices can be mobile and feature wireless communication, so the services they offer can appear and disappear on the network (as they connect and disconnect).
- Mobile devices can be battery powered and can be unavailable between the time the battery is exhausted and the time it is replaced.
- Devices might only be able to consume a fraction of the data related to a business transaction. Many elements of the XML would be irrelevant for a device.
- *Vice versa*, a device (e.g. a temperature sensor) cannot be expected to deliver the correct business context (e.g. the ID of its location in an asset management application) for the data it reports (this can be expected in an ERP system).

Although deploying web services on devices better embeds them in a company's SOA, shop-floor services are more fine-grained and often concentrate on technical issues. To bridge between the data types and semantics used at device level and those used at ERP system level, it will in most cases be necessary to use MES or SOA-based middleware acting as gateways.

As different manufacturers offer devices on the shop floor, they cannot be expected to have the same micro-controllers, have their communication application programming interfaces (API's) implemented in the same language, or run on the same operating system. However, a production process often requires devices from different manufacturers to work together. To make this possible today, shop-floor integrators have to invest heavily in device drivers and gateway mechanisms

that take time to develop, are expensive and are difficult to maintain. In the near future, if devices would use web services, a workflow process can be easily orchestrated using modelling languages; they can be easily changed, extended, and adapted to changing needs of the business. Following an SOA-based approach, machine functionality offered as services can be used as process steps and therefore easily orchestrated. Furthermore, different plant-level systems can gain access to data from the shop-floor.

Modern manufacturing networks often span across continents. Typical examples are semiconductor and automobile production. Both industries require a large number of production steps where some are executed by specialized companies, e.g., silicon wafers are etched and oxidized at one place whereas cutting and packaging is done at a different location. Sales and distribution are done by yet another partner while the design and IP on the product is often owned by a company located at another side of the globe. Bringing constant change on these highly optimized networks to adapt to market trends or to react to competition is a challenging task. It is the same difficulty faced when detailed monitoring has to be done on distributed, multi-party processes and resources, where the monitoring does not rely on manual data entry, but on actual, near real-time production and progress figures as they are reported by embedded software in the low-level devices of each partner. Existing supply chain platforms can hardly fulfil these requirements.

SOA on the shop floor offers such fine-grained, inter-organizational insight into production. Using standard modelling languages like BPEL or by modeling executable BPMN 2.0 processes in NetWeaver Composition Environment, workflows can be composed without any need for device drivers or knowledge about the underlying hardware. These workflows can be a part of a business process that interacts with business software or humans. Manufacturing consortia can now use shared shop-floor data, derive key performance indicators (KPIs) from business-to-business partners, and react to market trends faster than before. By analysing the securely shared shop-floor data and KPIs, the end-to-end process can now be globally optimized to achieve a better performance compared to local optimization at each participating entity.

The seamless integration of the shop-floor data into the collaborative production process also allows for flexible ways of handling critical events that can occur at the shop-floor at run-time, such as a machine breakdown. Such events can be made visible across the whole manufacturing chain among the business partners. Customers at the end of the chain can be informed about problems, e.g. delayed delivery or the cancellation of an order in a timely manner. This can reduce costs and enhance the long-term business relationships.

However, the example of error handling also reveals a strong requirement for successful adoption of the flexible production networks. For the participating organizations to be willing to share data that potentially reveals the structure and performance of their core operations, organizations either need to trust each other or one of them must have the power to dominate and dictate the conditions of col-

laboration. In both cases, secure and reliable communication between the entities is required.

14.5 Towards Dynamic Adaptation

The business world is highly competitive and to successfully tackle everyday's challenges, operational managers and executives demand high-dependability and wide visibility into the status of their business process networks. The latest is done usually via business KPIs. However, to react in a flexible and optimal way to changing conditions, real-time information must flow via all layers from the shop floor up to the business process level.

As shown in Figure 14.4, three different phases help us achieve the required level of flexibility:

- Sense: real-time monitoring over an event-based infrastructure for networked devices.
- Think: an autonomous decision support system (DSS) guarantees business continuity.
- Act: decisions taken from the DSS are enforced in the infrastructure.

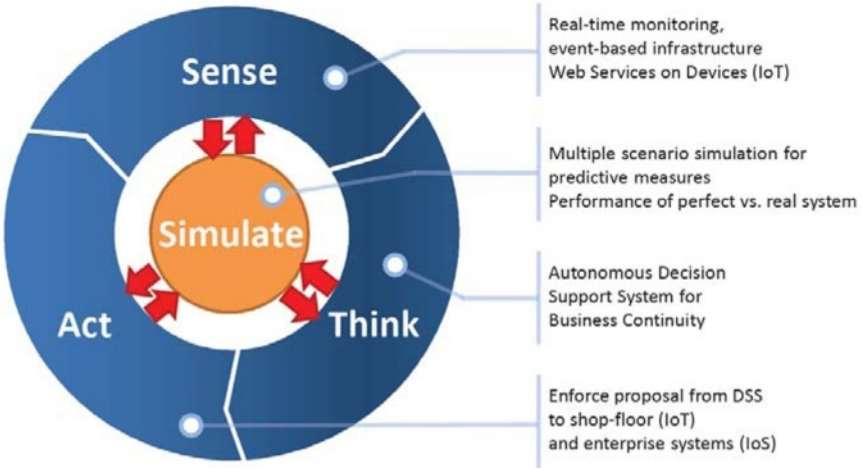

Fig. 14.4 Towards autonomous systems via cross-layer monitoring, simulation and management

In the centre of the approach is the simulation that provides predicted system behaviour that is continuously compared with real-world results. The key technical concept to the whole approach is a cross-layer communication in order to provide effective monitoring, simulation and management. In this approach we target two characteristics of an autonomous system:

- Self-healing: automatic discovery and correction of faults or possible preventive actions. The system can recover from well-known problems including those that can be dynamically identified based on the correlation of events (complex event processing).
- Self-optimization: automatic monitoring and control of resources of the system can be done, in order for the different components that recognize themselves in a goal-driven manner with respect to the environmental context they act on. In that case, early indicators can be correlated and emerging problems are easier to pinpoint.

Figure 14.5 depicts the architectural approach proposed in this chapter. The core idea is the dynamic connection of enterprise systems with the shop-floor assisted by a DSS.

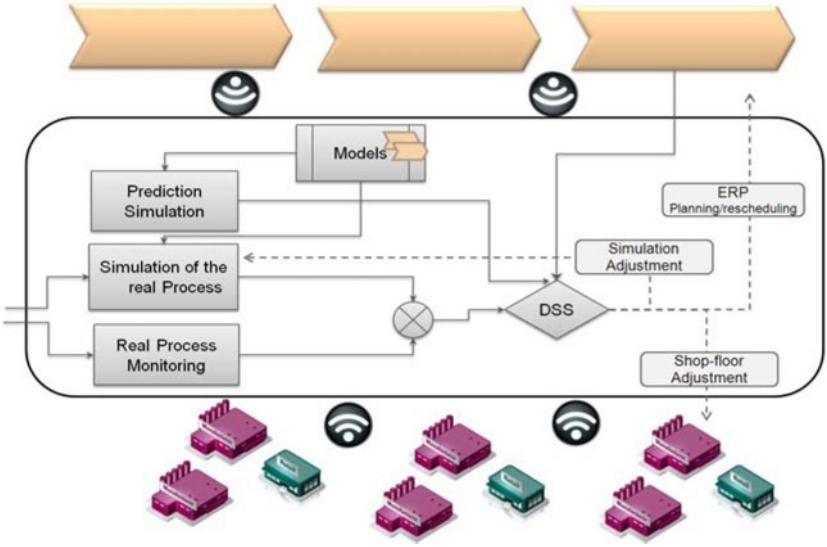

Fig. 14.5 Overview of the reactive system

The DSS takes into consideration dynamic data coming from monitoring the shop floor, running simulations and its results are given as input to the business process control, the shop-floor and even on fine-tuning the simulation itself. In this concept, continuous real-time data flow into a monitoring system. In parallel, a model of the shop-floor executes in a simulator. At specific intervals depending on the time or tasks, the output of the simulation and the monitoring are evaluated. Any deviation σ is used as input from the DSS. Parallel to σ, the DSS considers inputs from the enterprise systems as well as the prediction simulation, which predicts the next system state(s) of according to the existing models the system will continue to perform in the same way. The DSS considers all the input and makes decisions, e.g., for optimizing the performance of the system, preventing faults

that would happen if the mode of operation is unchanged, etc. The DSS decisions are fed as input to the business systems, the shop-floor, and the simulation itself so that their behaviour can be adapted. Using precise information on the problems occurring (or predicted to occur) on the production line, the DSS can define measures to automatically modify the business process to heal the system. The result is that we are moving towards a self-* system that monitors and adapts itself according to the evaluation of the input from the sources mentioned.

14.5.1 Simulation

Simulating a process workflow as a production model on a computer takes away the risks of heavy investment. Modelling tools such as flowcharts, process mapping, and spreadsheets are often used to identify how the shop floor would look like for a particular business objective. However, such tools only show the relationships between processes and do not generally provide any quantitative performance measures. They are static and deterministic and do not consider the dynamics of real-life work in progress.

Dynamic production model simulation tools like WITNESS (Ahmed *et al.*, 2008) and ProModel (www.promodel.com) consider the dynamic characteristics of production, e.g. process flow, processing times, setup requirement, labour, control rules, breakdown, shift, loading schedule, etc. We propose to go one step further by using the simulation results and comparing them to the live input coming from the shop floor in order to assess the situation, proactively determine problem zones, and optimize the shop-floor.

Figure 14.6 presents the generic states of a machine in a production line. Based on a known machine state, a deterministic action can be performed. Nevertheless, defining the current state of the machine can be a challenging task. The information available to the back-end system consists of a great amount of events triggered through the assembly process. They could be identical to the simulated set of events and states, or deviate from the simulated conditions. Therefore, it is necessary to process these events in order to identify patterns that indicate which is the current state of the production line.

The advantage of production model simulation tools is that by changing the characteristics of production, the results on the shop floor can be accurately determined (Pidd and Robinson, 2007). These tools give a good in-depth understanding of how the manufacturing process would react to different situations on the shop floor. Such results can be considered as a reference for a series of deterministic characteristics of the shop floor (Ahmed *et al.*, 2008). As depicted in Figure 14.5, we propose a methodology where the workflow of a particular production process is continually monitored and compared to (pre)simulated results.

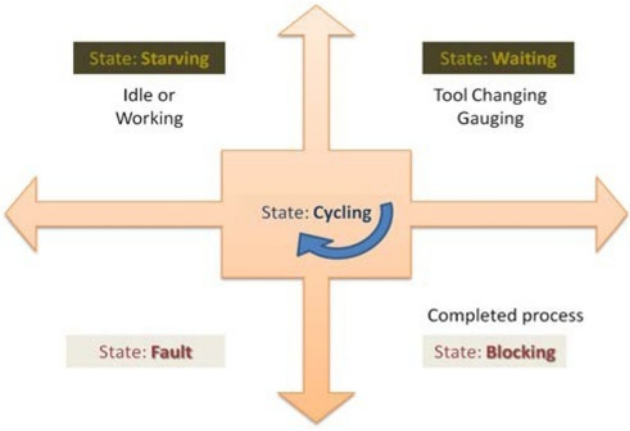

Fig. 14. 6 Machine states

The state of the actual workflow process is continually monitored and compared with the simulated result on the shop floor. Based on the possible deviation σ, the workflow process or the business process can be affected. The deviations of the shop floor behaviour from the expected result of the simulation have to be categorized in order to determine if the current condition would be tolerable or it would lead to a critical bottleneck. Various algorithms and methodologies can be combined to identify and categorize the state of the current production line.

14.5.2 Self-healing Mechanisms

Through a comparison between the expected state (simulation results) and the real state of the shop floor, it is possible to identify malfunctions in the system. This information is essential for the system to self-heal.

As an example, consider a knitting factory where T-shirts are manufactured through sequential steps like cutting, assembly, buttoning, quality check, and packaging. For instance, as shown in Figure 14.7, if the machine responsible for executing the process "Cutting 1" fails, the production order of the T-shirts with quality 3 will stop. This can delay the order delivery, cause reputation loss, and even breach a contract, which implies high costs to the factory.

If the manufacturer defines a maximum cost for the T-shirt production, it is possible to re-map the business process to avoid a production halt considering the new state of the system. This process re-mapping is depicted in Figure 14.7. With the modification in the business process, the system self-heals and continues the production, while a maintenance workflow is triggered. Although the item cost increases, it remains within the threshold specified by the manufacturer, and prevents major losses due to production halts.

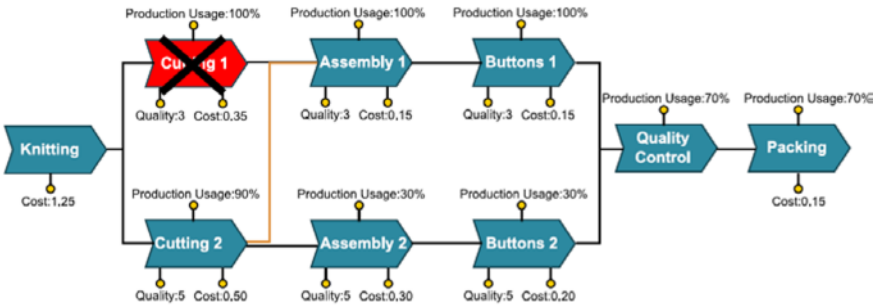

Fig. 14. 7 Self-healing and self-optimizing process re-route

Another self-healing mechanism investigated by this project explores predictive maintenance and possible production bottlenecks. Based on real-time data from the shop-floor and having identified the current status of the production line, it is possible to predict the course of the current production. This is done based on analysis of the previous production history and also based on the simulated production model. The result of such prediction model analysis is forwarded to a DSS, which then reacts by providing input to the business process modeller (Figure 14.5). Finally, the integration of ERP business processes and such DSSs can be performed through the SAP Manufacturing Intelligence and Integration (SAP MII) tool.

14.5.3 Self-optimizing Mechanisms

Business processes are available as services in the enterprise service repository. Hence, a set of rules can be modelled in the DSS to invoke a corresponding business process at the prediction of a critical bottleneck state on the shop floor. Alternatively, shop-floor devices hosting web services can be effectively used in such scenarios to prevent malfunction or breakdown of the machine. The DSS can reduce the production cycle on a particular assembly line when the system foresees a non-linear increase in temperature or a production variable on the workflow.

We propose to base the optimization process on swarm-intelligent (SI) principles (Bonabeau *et al.*, 1999). These methods were originally inspired by observation of various natural phenomena, in particular the collective behaviour of social insects and flocking and schooling in vertebrates. The application of SI to distributed, real-time, embedded systems aims at developing robust task-solving methodologies by minimizing the complexity (including the intelligence) of the individual units (in our case machines of the assembly line) and emphasizing parallelism, and self-organization. From an engineering standpoint, the principal advantages of SI system design are four-fold: scalability, from a few to thousands of units; flexibility, as units can be dynamically added or removed without explicit

reorganization; robustness, not only through unit redundancy but also through an adequate balance between explorative and exploitative behaviour of the system, and simplicity (and low-cost) at the individual level, which also increases robustness. These properties would be highly beneficial if applied to machine production lines, and could be further optimized when machine have access to global information about the whole manufacturing process.

In particular, we propose to use threshold-based algorithms (TBA) for a flexible task allocation mechanism to decide of the dynamic path in the production line. TBA have been initially used to model the dynamic task allocation decision process in ant colonies, and has been successfully applied for example for power-aware optimized load balancing (Cianci *et al.*, 2005). Using TBA at the production-line level enables a reactive and fully decentralized decision process done dynamically by the machines at run-time, based both on the objects to process, external data (environmental data, priority of the tasks, market values, etc.), and the proprioceptive data of the machines. TBA model group behaviour based on a small number of control parameters (thresholds) that affect whether or not a particular task will be executed by a given machine. For this, every machine has an internal threshold value which is a function of different dynamic and static factors (price and time associate with task execution, machine current state, etc.). Each task to process will have its own stimuli value that will be compared with the threshold of the machines and will be used to decide which machine will perform the task. Thresholds are allowed to change and become heterogeneous over time as a function of stimuli encountered and tasks performed, and this can lead to specialization and division of labour.

14.6 Concept Validation in Prototypes

We consider some general use cases to illustrate of shop-floors with ERP systems. Status of machines on the shop floor is usually monitored. This information could represent completion of an order, number of work pieces produced or even failure of a machine, which is vital for consecutive production plans.

The main common characteristics we focus on are:

- *Smart devices*: shop-floor machines and other sensors, PLCs, and IT devices are the actors forming an “internet of things” in the factory. They all offer their functions (e.g. start/stop, swap to manual/automatic mode) or status (e.g. power consumption, mode of operation, usage statistics, etc.) through web service interfaces, either directly or through the use of gateways or mediators.
- *Business logic and visualization services*: In our prototypes, the business logic services are supported by a service composition engine and a graphical user interface (GUI) using a visualization toolkit. An operator can use existing tools to create the business rules. Via visualization tools the plant-floor status and the

overall process execution can be analysed in detail. As an example, the operator can instantiate and use a set of widgets such as gauges and graphs to monitor in real-time the status of production machines and associate it with the respective orders or business effects.

- *Enterprise applications*: This refers to high-end business software such as ERP or product life cycle management. The idea at this level is to visualize processes rather than the machines executing the processes. This layer is connected to the plant-floor devices through the other layers. As such it can report machine failures and plant-floor information on the process visualization and workflow. Furthermore, business actions (e.g. inform customers about a possible delay) can be executed based on this timely information.

14.6.1 Machine Monitoring, Dynamic Decision and Order Adaptation

On a production line we need to monitor a robotic gripper for overheating as this would cause further malfunctions. As shown in Figure 14.8, a SunSPOT wireless sensor node is attached to the gripper and checks continuously the temperature at the relevant location right before the gripper starts its operation, e.g., open or close.

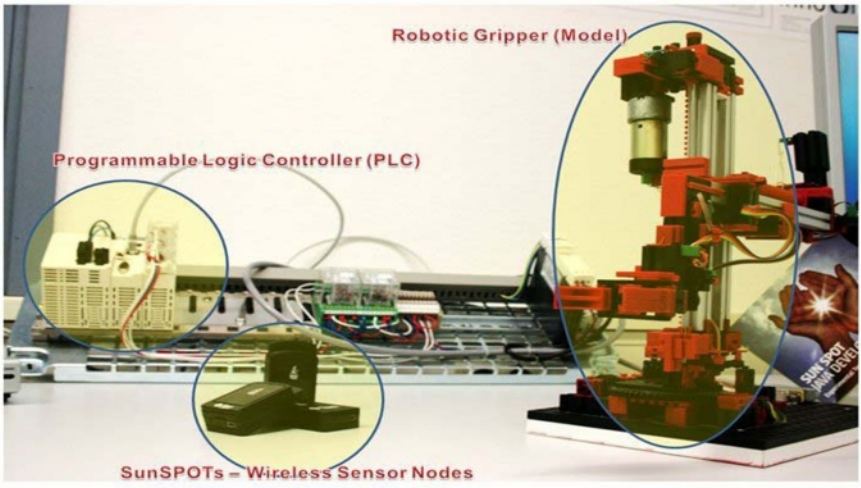

Fig. 14.8 Overheating monitoring via wireless sensor and ERP-supported control of a process

A PLC controls the robotic gripper and offers its available functionality as several web services; the same holds true for the wireless sensor. In SAP MII we have

modelled the business logic (shown in Figure 14.9), which takes decisions based on the data from the robotic gripper and the wireless sensor. The operation of both devices generates events, which are picked up and consumed by SAP MII as we extended the tool to subscribe to these.

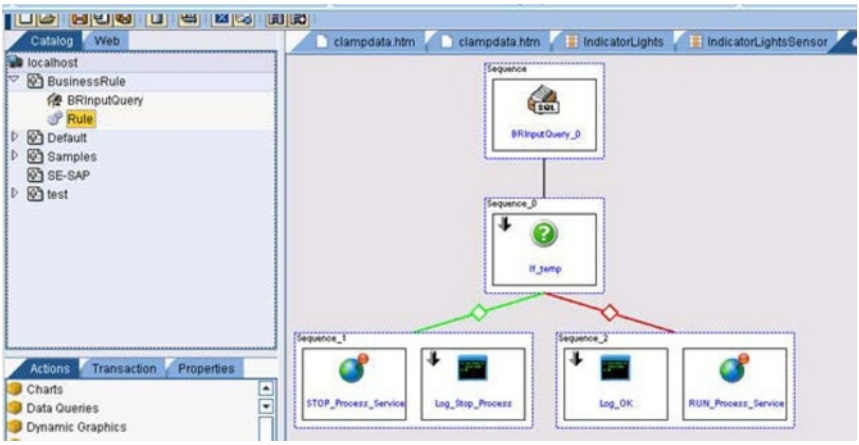

Fig. 14.9 Modelling business logic in SAP MII

In our scenario, during normal operation the robot gripper should not exceed a specific temperature limit. If the threshold is exceeded, a business rule triggers necessary countermeasures, e.g., it stops the gripper in order to prevent damage and invokes appropriate enterprise-level services. This includes visualizing the stopping of the gripper and possible delays in production, e.g. by changing the colour in the management view of the factory as depicted in Figure 14.10. If the resulting order delay is critical, a notification is generated for the key account manager, about the fact that an order for his client is in danger of missing the deadline. The timely information about production delays has the potential to prevent lost sales, which are more likely to occur if customers do not receive their order in time and are not informed promptly and reliably about it. The management cockpit of Figure 14.11 is another example of web service composition at a higher layer where we integrate the real-time status of the factory with Google Maps in order to visualize the overall effect of a single failure that results from an order delay.

Apart from the high-level view for the manager, there is also other visualization information for the operator. A simple gauge fed with the temperature data provided by the wireless sensor can be depicted also in real-time via the SAP MII tool. For this purpose, he uses manufacturing intelligence software and displays the gauge on a screen situated close to the robot. Finally, the sales manager can also benefit from the SOA paradigm, as, e.g., the output of the business rule is connected to an ERP system, which provides up-to-date information about the execution of the current orders. Whenever the process is stopped because the rule

was triggered, an event is sent to the ERP system through its web service interface. The ERP system then updates the orders accordingly and informs the clients of a possible delay in the delivery.

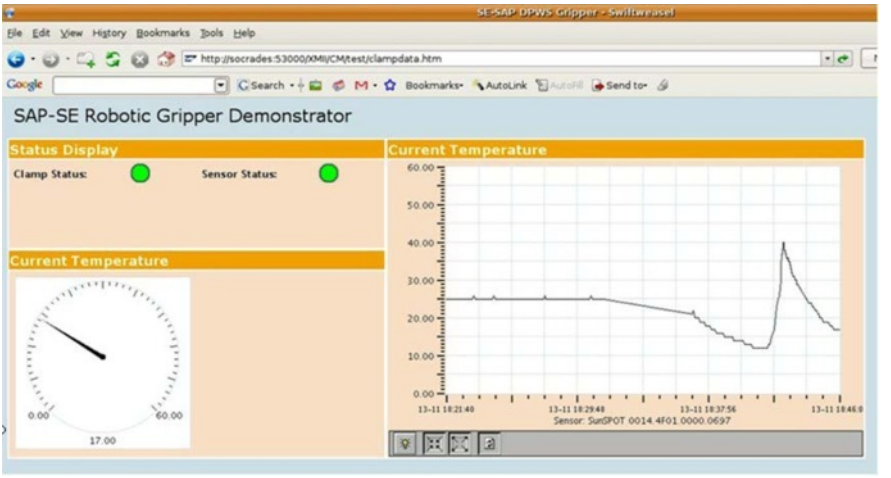

Fig. 14.10 Live reporting in SAP MII of the current shop-floor status based on events received from the devices

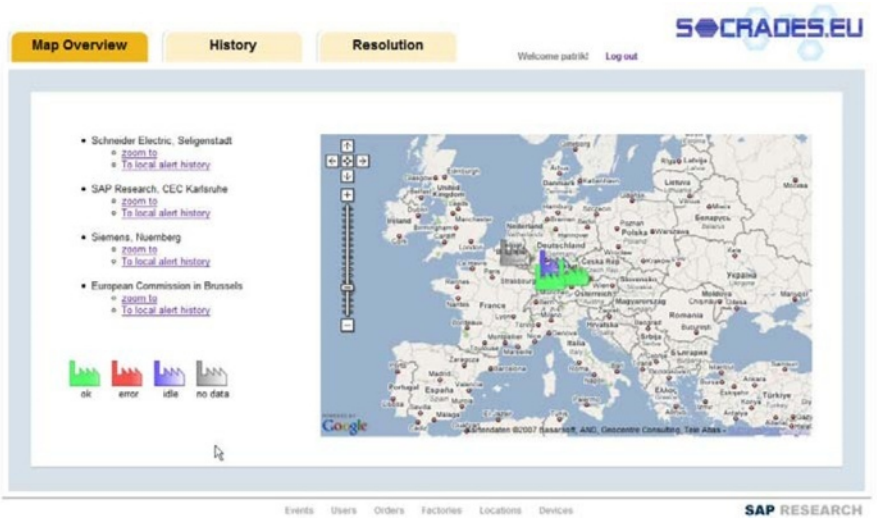

Fig. 14.11 High-level factory view in the management cockpit

14.6.2 The Future Shop Floor: Mashup of Heterogeneous Service-oriented-architecture Devices and Services

Today, integrating devices in applications requires not only advanced knowledge of the device, its configuration and the way it connects, but also the installation of highly specialized software that glues the data (often in proprietary format) with applications. Such an integration model is costly, application specific and creates isolated islands for each shop floor. As a result, it is extremely hard for enterprise service developers to enrich service functionality with real-time data coming from the shop floor. The SOA concept has proven very successful for gluing heterogeneous systems, and if the same would be applicable for devices this would be a significant step forward in the direction of coupling the real-world and the business world.

To demonstrate the concept we have taken several devices (some of them IP-enabled) and wrapped their functionality with web services. More specifically as depicted in Figure 14.12, we have:

- An RFID reader: RFID tags as they appear are read by the reader, which raises events showing info with respect to the tag. Each tag is considered to be integrated with a product and serves as a token that links it with the business information (e.g. order).
- A robotic arm: the functionality of the robotic arm controller has been wrapped, e.g. grab or move.
- An IP electricity switch: an alarm lamp has been attached in an IP electricity switch, which offers the on/off functionality as a web service.
- A wireless sensor-controlled emergency button: in the IO pins of the SunSPOT wireless sensor, the emergency button has been attached. The press or release of it is captured by the SunSPOT, and a web service event is generated.
- A wireless sensor for vibration monitoring: the capability of SunSPOTs to measure acceleration is used for vibration monitoring via the sensor mounted on the robotic arm. This monitors the transportation conditions of the product by the robot to make sure it adheres to the quality guidelines for the specific product.

Production parts that arrive on the factory shop floor are equipped with RFID tags. As they reach the processing point, their data is read by an RFID reader. An event is generated by the reader with all the necessary data, e.g. RFID number, and placed on the network. The robot gets notified by this event and picks up the production part. By matching data from the enterprise system and the RFID tag, it knows how to further process the part.

In parallel, the SunSPOT mounted on the robot monitors the vibration and if this exceeds a specific threshold an event is raised to immediately stop the process. The same holds true for the emergency switch, if an operator at the shop floor for any reason wants to immediately stop the process, s/he presses this switch. The

result is captured by the wireless sensor and an event is raised to immediately stop the robot.

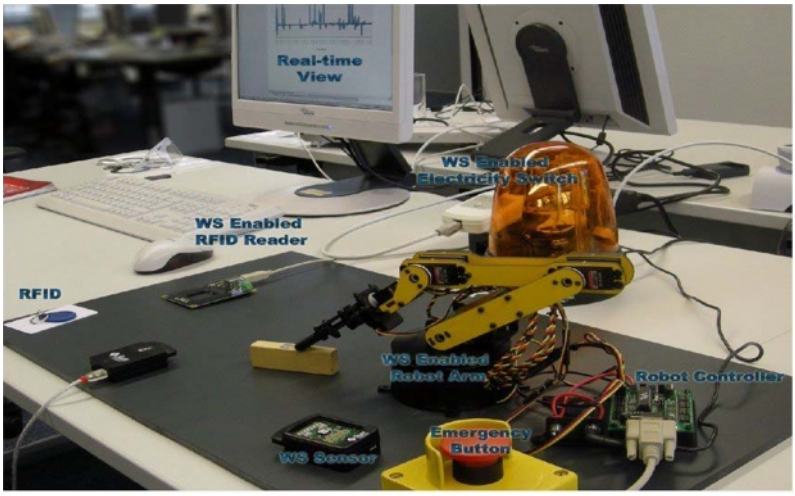

Fig. 14.12 Web-service-enhanced/wrapped heterogeneous devices - RFID readers, sensors, robot, IP electrical switch, etc

Once an event occurs, the devices process it and react accordingly. As such, the robot picks the emergency shutdown event and immediately stops its action. Also the IP electricity switch turns on the emergency light once it receives the event. At a higher level, there is an enterprise application for shop-floor reporting called SAP Business Objects. There we have a real-time monitoring of the shop-floor status as the application also subscribes to all events raised. The plant manager can immediately see (Figure 14.13) the status of the ERP orders, the production progress, and the device status, and have a global view of all factories and the possible side-effects of a production line delay due to shop-floor device malfunctions.

All of the communication is done via web service technologies and the realization of this specific scenario was made possible by having a composition of the available functionality that all devices and applications expose as a service. In legacy systems, integration of a new device or reassignment of its role would result in reconsidering how the device integrates with other devices and how they control it. However, with the SOA approach described, a new scenario is possible by modifying the orchestration of the services already available.

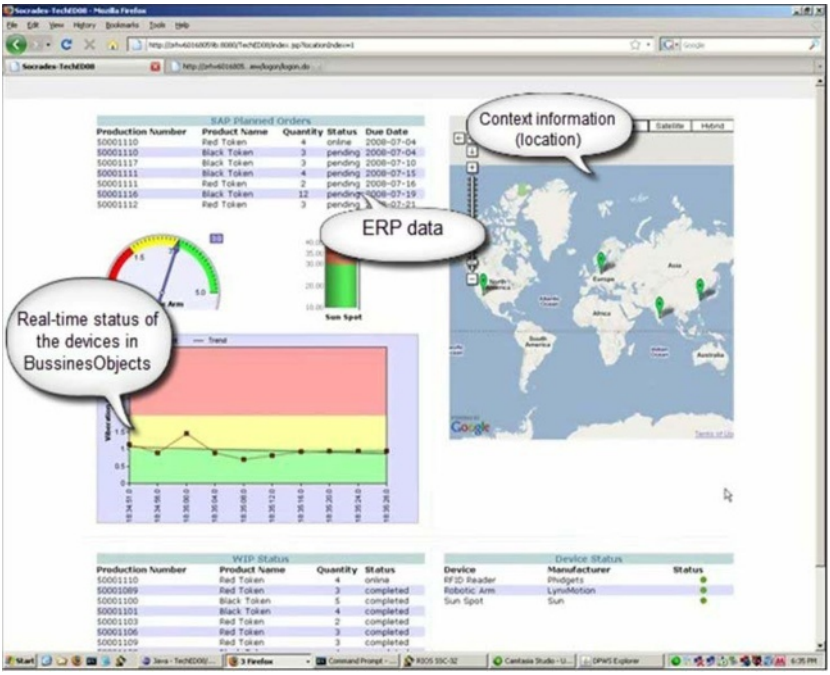

Fig. 14.13 Real-time reporting via enterprise visualization tools

14.6.3 Dynamic Supply Chain Management Adaptation

The SAP supply chain management (SCM) production planner (PP) system supports manufacturing by optimizing throughput times and bottleneck capacities using new scheduling processes (supply optimization) and by achieving online integration of production planning and control activities. This provides a transparent overview of the entire order network. For an optimized schedule in the production plan, the SAP SCM PP needs data about the available resources from the shop floor to cross boundary locations.

The quality and validity of the information is important for real-time simulation of the shop-floor resources. To achieve this, SAP has added MII as an integration middleware between the MES on the shop-floor and the SAP ERP systems. MII also offers bidirectional connectivity between the control, field-level systems and the ERP landscape. As a result, the plant manager could have an overview of actual orders getting executed on a particular machine and what is the status of that machine. Moreover intelligent information can be extracted from manufacturing analytics and performance factors provided by the SAP MII.

In this prototype we have used a test rig that consists of a suite of pneumatic-based hardware that does operations like picking, placing, moving, drilling, proximity sensing and stocking. The overview of the hardware is pictured in Figure 14.14. The test rig is supported by a 10-bar compressor and operates at 24 V DC from the power supply unit. It is assumed that it produces tokens, which are drilled with a hole in the centre. The tokens are supplied from a magazine. The rig drills a hole on the token, checks its colour and material, and sorts it to the corresponding storage magazine.

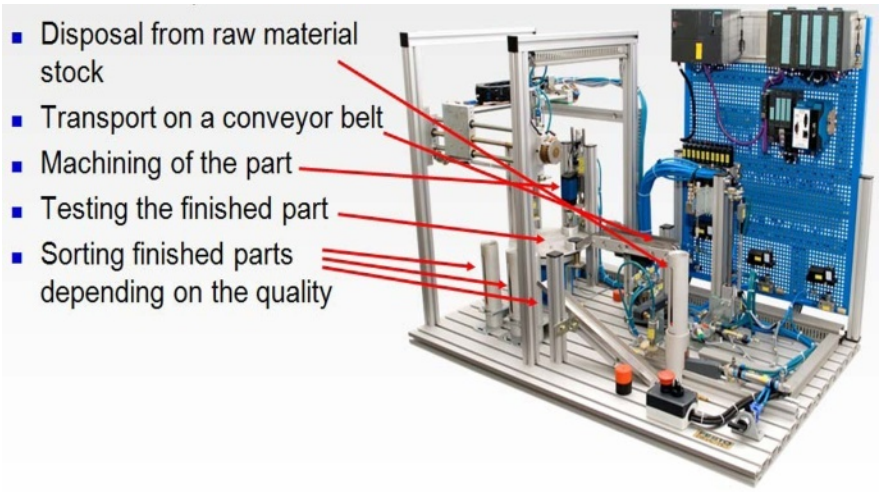

Fig. 14.14 Main operations

The main parts of the test rig (also seen at Figure 14.15) are:

- Input magazine and distributor: this module consists of a shoving-out cylinder operated pneumatically, a pile magazine, which stocks the red and black tokens and a swivel arm.
- Proofer: the proofer module consists of the proof station, a rejection cylinder and a conveyor.
- Rotating work table: this module consists of a turn table, a drill press with clamping cylinder and a piston that checks for a hole in the token called the proof cylinder.
- Dispatcher: the dispatcher reads the colour information from the PLC and calculates the trajectory of the vertical and horizontal motion of the linear actuators, picks up the component and moves the actuators towards the storage magazine where the tokens are stored respective to their colour.

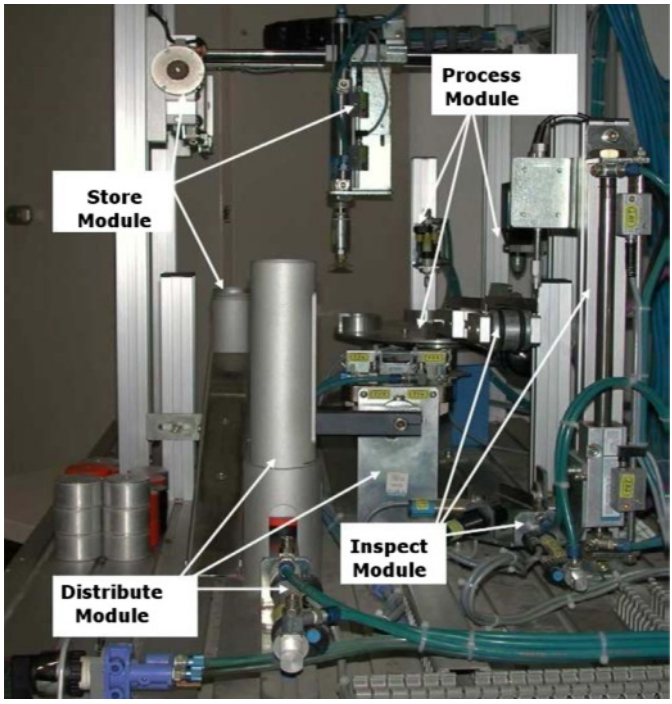

Fig. 14.15 Material flow

All these modules are controlled by one PLC. The rig uses the Siemens S7-300 PLC with an OPC DA interface to communicate to an execution system. In this demonstrator the execution operations on the PLC are wrapped with web services in a middleware component called the mediator. The mediator monitors data within the Siemens S7-300 PLC using the OPC interface. It also controls the start and stop operations of the PLC. A Mediator exposes the start and stop operations as a web service. These operations can be invoked from the SAP MII software. The mediator retrieves the status information from the PLC. It propagates this information as web service events through DPWS and is also visible via enterprise applications such as the SAP MII. At the end of each operation, there are events generated by the mediator to SAP MII. Either the successful completion of the operation or the failure of a task is reported. To that extent, we have used DPWS to connect the enterprise software and test rig via the mediator.

The SAP MII software performs the basic interconnection between the shop floor and the ERP landscape. SAP MII transactions analyse the results using complex business rules, which are developed using the MII business logic editor. The rules check which token (red or black) is produced and check against the corresponding production order. If there are more red tokens produced than the ones required in the production order, they are stored and then logically deducted in the next production plan, which is already designed and ready to be executed, and in

the pipeline. This does not increase the production time, rather neutralizes the delay. The MII also shows the status of the test rig modules. Figure 14.16 presents the snapshot of the SAP MII GUI that is visible to the plant-floor manager.

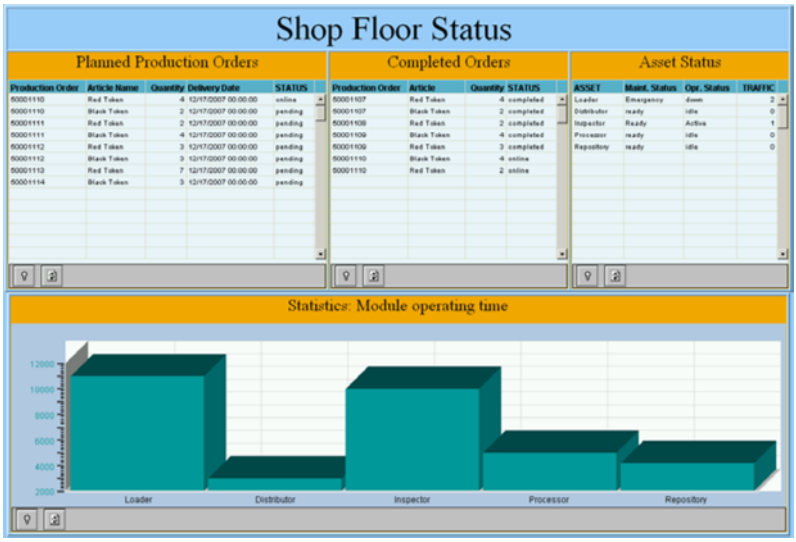

Fig. 14.16 Plant manager's view based on SAP MII

14.6.4 Taming Protocol Heterogeneity for Enterprise Services

When enterprise application designers develop new features, taming the heterogeneity can be a significant burden. The SIA architecture offers a pluggable framework with the aim to abstract the functionality from the protocol itself, at least for business-related messages that are valuable to the enterprise system. As such, business applications can be more flexibly developed, while the translation of messages for a specific protocol is pushed down to the device level.

In this prototype we considered a solar cell manufacturing plant which produces thin-film photovoltaic cells. Manufacturing such semiconductor devices is similar to producing integrated circuit chips. Semiconductor manufacturing involves complex and machine intensive workflow processes. This means, machines from different manufacturers would need to exchange large amounts of data intensively to complete a production process.

Since its specification, the SEMI Equipment Communications Standards (SECS) has been widely used on the shop floor in a semiconductor fabric (Cimetrix, 2009). SECS-II/HSMS offers the possibility of delivering SECS messages over IP, but only between two points; no multi-point communication is possible-a

drawback for most internet-technology-based applications. The thin films are transported between the processing machines using conveyor belts. We consider such a conveyor system that is controlled by an OPC UA-based PLC (e.g. Beckhoff CX1010). OPC UA-based manufacturing devices are common on the shop floor like this one. We take the example a step further by monitoring the energy consumed by the whole shop floor and estimate the energy consumed to produce one photovoltaic cell. We add a smart meter to the production facility, periodically record the energy consumption data, and associate the energy costs from an ERP system. The smart meter would send data using REST-based HTTP packets (see Figure 14.17) to the enterprise application.

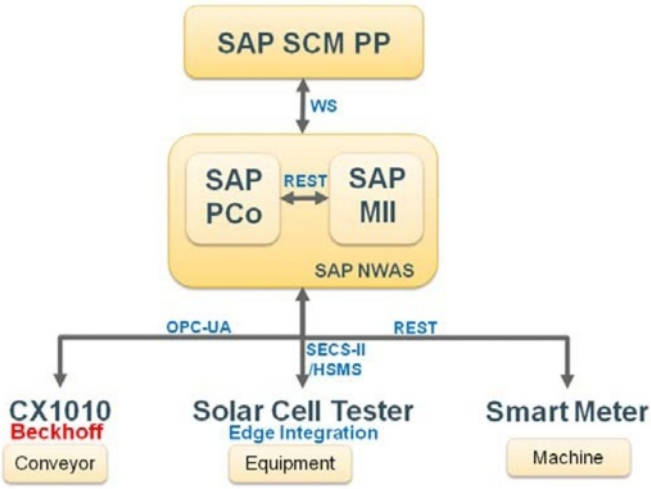

Fig. 14.17 Multi-protocol interaction with web services, REST, OPC-UA and SECS-II/HSMS

Traditionally SECS-II/HSMS messages are sent only from the equipment to the host interface, which is the MES system. Providing alternative connections or multiple communication endpoints is rather expensive and requires additional software development that can be tedious to maintain when the features of the manufacturing equipment changes. When two entities are trying to connect to SECS host equipment, the connection is terminated to either one of them for a brief period; if not, multi-point communication is supported. As a consequence, challenges arise when trying to make the processes adaptive or trying to extend it to additional destinations. Introducing variables to adapt to the dynamic nature of the shop floor is very expensive for companies that span multiple production locations and several heterogeneous IT systems. We propose to solve this problem by extending the SECS-II/HSMS with web services. Combining web services with SECSII/HSMS enables semiconductor manufacturing equipment to report alarms, events and control signals to a variety of IT systems in the plant directly in real time, avoiding expensive gateway connectors.

With new and powerful devices on the shop floor, manufacturing equipment can deliver real-time data to multiple destinations. As a result, a mashup of real-time data analysis can be done that can lead to estimation of trends from current production. In this prototype, the energy consumed to produce a photovoltaic cell can be estimated using web services on traditional protocols and REST-based light weight frameworks. Simultaneously, standard XML-based technologies like OPC UA can be mashed up from the device level, offering a new level of energy-aware production.

14.6.5 Energy Monitoring and Control via Representational State Transfer

Interacting with devices in many cases should be done very easily, in a short time and with no strict guarantees or need for a full-blown functionality. As such, heavyweight web service integration in specific scenarios might not be the best option; on the contrary, REST integration is a better match for lightweight integration. In most early web-based approaches, HTTP is used only to transport data between devices, while in fact HTTP is an application protocol. Projects that specifically focus on re-using the founding principles of the web as an application protocol are still not common. Creation of devices that are web-enabled *by design*, would facilitate the integration of physical devices with other content on the web. As pointed out in Guinard and Trifa (2009) and Wilde (2007), REST-enabled devices would not require any additional API or descriptions of resources/functions.

The architectural principle that lies at the heart of the web, namely REST as defined by Roy Fielding (Fielding and Taylor, 2002), shares a similar goal with more well-known integration techniques such as WS-* web services (SOAP, WSDL, etc.) or DPWS, which is to increase interoperability for a looser coupling between the parts of distributed applications. However, the goal of REST is to achieve this in a more lightweight and simpler manner, and focuses on resources, and not functions as is the case with WS -* web services. In particular, REST uses the web as an application platform and fully leverages all the features inherent to HTTP such as authentication, authorization, encryption, compression, and caching. This way, REST brings services “on the web, directly from the browser”: resources can be linked and bookmarked and the results are visible with any web browser, with no need to generate complex source code out of WSDL files to be able to interact with the service.

Smart gateways can be used to integrate or abstract from the protocol heterogeneity. These implement in principle the lowest layers defined in SIA. As an example we have created a gateway that runs a web server, and understands the proprietary protocols of different devices that are connected to it through the use of dedicated drivers. As an example, consider a request to a sensor node coming from the Web trough the RESTful API. The gateway maps this request to a re-

quest in the proprietary API of the node and transmits it using the communication protocol the sensor node understands (e.g. Zigbee www.zigbee.org). A smart gateway can support several types of devices through a driver architecture as shown in Figure 14.18, where the gateway supports three types of devices and their corresponding communication protocols. Technical details of the Smart Gateways can be found in Guinard and Trifa (2009) and Trifa *et al.* (2009).

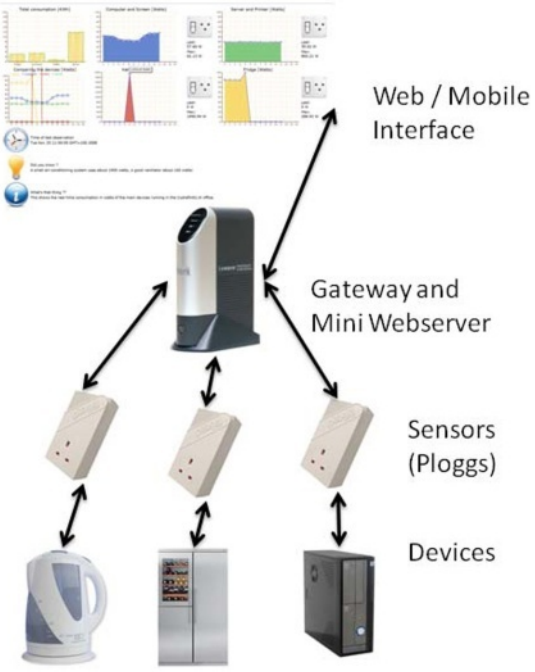

Fig. 14.18 Device integration via a gateway

In order to empirically analyse and test the potential of the RESTful approach for real-world services and how our approach could become the basis for the web of things, we implemented the architecture on two WSNs platforms: the SunSPOT sensor nodes (www.sunspotworld.com) and the Ploggs energy eensors (www.plogginternational.com). The Sun SPOT platform is a wireless sensor node particularly suitable for rapid prototyping of WSNs applications. SunSPOTs run a small-footprint Java Virtual Machine that enables the nodes to be programmed using the high-level Java programming language (Java Micro Edition CLDC java.sun.com/javame). The RESTful architecture we designed and implemented for the Sun SPOTs is composed of two main parts: a software stack embedded on each node, and a proxy server to forward the HTTP requests from the web to the SPOTs. The Ploggs smart gateway is a component written in C++ whose role is to

automatically find all the Ploggs in the environment and make them available as web resources. The gateway first discovers the Ploggs on a regular basis by scanning the environment for Bluetooth devices. It then filters the identified devices according to their Bluetooth identifier. The next step is to make their functionalities available though simple URLs, and for that a small-footprint web server is used to enable access to the sensors' functionalities over the web. This is done by mapping URLs to native requests on the Plogg Bluetooth API. Hence, requesting the energy consumption of a Plogg can be done simply be issuing a GET request on the following URL, as suggested by the REST architectural principles (Fielding and Taylor, 2002): to http://webofthings.com/energymonitor/ploggs/conveyerbelt. This returns a JSON (JavaScript Object Notation) json.org/ document containing information about the energy consumption of the conveyor belt. Note that JSON is an (compatible) alternative to XML often used as a data exchange format for web mashups. Since JSON is a lightweight format we believe it quite adapted to devices with limited capabilities.

The idea of the energy visualizer prototype built on top of the Ploggs Smart Gateway and the RESTful SunSPOT is to offer a dashboard user interface on the web to control and monitor the energy consumption at the device level. It offers six real-time and interactive graphs. The four graphs on the right side provide detailed information about the current consumption of all the appliances currently in the vicinity of the smart gateways. The two remaining graphs show the total consumption (kWh), and respectively a comparison (on the same scale) of all the running appliances. Finally, a switch button next to the graphs enables one to power on and off devices over the web.

This dashboard is built as a mashup that uses the RESTful Plogg API in a Google Web Toolkit (GWT) application code.google.com/webtoolkit/. The GWT is a powerful platform for building web mashups since it offers a large number of easily customizable widgets. For the graphs shown on Figure 14.19, we use the Open Flash Chart GWT Widget Library. This library offers a comprehensive set of graph widgets.

The ambient meter prototype is implemented on a RESTful SunSPOT which uses a Ploggs smart gateway for gathering the energy consumption of any place it is located in. It uses an HTTP connector we implemented in the RESTful Sun SPOTs to contact the Ploggs smart gateway. Every 5 seconds, the SunSPOT will poll the following URL using the GET method: http://localhost/energymonitor/load. The SunSPOT changes its colour according to the energy consumption of the place it is currently located in. Thus, by combining these two services, the SunSPOT is turned into an ambient meter that can assess the energy consumption of the place it is located in, demonstrating the ease of integrating.

Fig. 14.19 Energy monitoring per machine

14.7 Discussion and Future Directions

We have presented how to deploy web services on shop-floor devices to connect them to enterprise systems. With web services, enterprise systems are also able to subscribe to events and take advantage of real-time information flow for optimization of the production planning or reaction to unexpected changes. The clear advantage of pushing SOA concepts down to device level is that business application developers can design and implement new functionality in enterprises without focusing on the devices but on the services they provide. As such, another abstraction layer based on well-known and used web service standards will ease the integration of enterprise and shop-floor systems. The scenario depicted in this chapter is a proof of concept for this easier and tighter integration. The results show that the dynamic nature of the shop floor can be utilized efficiently to plan further production orders and even implement last-minute changes on the production line using real-time data (real-time reconfiguration based on the application needs). As all communication is done via web services (in our case DPWS) it is easy for other entities (whether they are services or devices) to subscribe and get the necessary info while in parallel being agnostic to the actual implementation details. This greatly increases interoperability and reduces costly integration time.

In our next steps we plan to further work on better web service integration, also with the usage of OPC UA as well as improvements regarding fulfilment of typi-

cal industrial automation-specific requirements like real-time constraints, reliability, safety, or security. Special attention will be paid to improve easy configuration of integration components like gateway or mediator based on device description technology known within industrial automation. This will pave the way for auto-configuration on the gateway and mediator level and enhance integration of legacy systems into higher-level management applications.

14.8 Conclusions and Future Work

Ubiquitous, SOA-based device integration leading to interaction of devices with enterprise services in a timely manner is an important vision that creates substantial impact both from the perspective of research and industrial application. The paradigm of services on every layer of the network will influence the structure and operation of future intranets and the future internet. Dynamic service discovery and composition will enable a new generation of applications that will more closely couple physical environments and processes with the corresponding models in business software, which are their virtual counterparts.

We have presented in this chapter the imperatives and motivations for more dynamic and flexible production lines in the factory of the future. With increasing competition – but also collaboration – around the globe, more information exchange and cross-layer communication is becoming necessary. We have introduced technologies that will be found in the future factory. They will entail more dynamic and adaptive production equipment that will closely collaborate with each other through real-world services and adjust its behaviour dynamically. We described several prototypes that demonstrate our ideas and depicted the applicability of the concept.

To continue our work, we will further investigate collaborative intelligent behaviour and improve its stability over distributed infrastructures composed of large numbers of heterogeneous networked embedded devices. Allowing for a flexible, semantic discovery of services is also a promising topic for further research, just as providing a rigid security framework for the described environments.

Acknowledgements The authors would like to thank the European Commission and the partners of the European IST FP6 project "Service-Oriented Cross-layer infRAstructure for Distributed smart Embedded devices" (SOCRADES - www.socrades.eu), for their support.

References

Ahmed R, Hall T, Wernick P et al. (2008) Software process simulation modelling: A survey of practice. J. Simul., 2:91–102

Bonabeau E, Dorigo M, Theraulaz G (1999) Swarm intelligence from natural to artificial systems. Oxford Univ. Press, New York, 1999

Chan S, Kaler C, Kuehnel T et al. (2005) Devices profile for web services. Microsoft Developers Network Library

Cianci C, Trifa V, Martinoli A (2005) Threshold-based algorithms for power-aware load balancing in sensor networks. In: Proceedings of 2005 IEEE Swarm Intelligence Symposium, IEEE-SIS, Passadena, CA, USA

Cimetrix (2009) SECS/GEM SEMI standards overview, http://www.cimetrix.com/gemintro.cfm

Colombo A W, Karnouskos S (2009) Towards the factory of the future–a service-oriented cross-layer infrastructure. ICT shaping the world, a scientific view. ETSI, John Wiley and Sons Ltd

de Souza LMS, Spiess P, Guinard D et al. (2008) Socrades: A web service based shop floor integration infrastructure. In: Proceedings of Internet of Things 2008 Conference, Zurich, Switzerland, 26–28 March

Dickerson R , Lu J, Whitehouse K (2008) Stream feeds: an abstraction for the world wide sensor web. In: Proceedings of the 1st Internet of Things Conference (IOT), Zurich, Switzerland, 26–28 March

Drytkiewicz W, Radusch I, Arbanowski S et al. (2004) A REST-based protocol for pervasive systems. In: Proceedings of the IEEE International Conference on Mobile Ad-hoc and Sensor Systems, pp. 340–348

Fielding RT (2000) Architectural styles and the design of network-based software architectures. PhD thesis, University of California, Irvine, Irvine, California, USA

Fielding RT, Taylor RN (2002) Principled design of the modern web architecture. ACM Trans. Internet Techn., 2(2):115–150

Fleisch E, Mattern F (2005) Das Internet der Dinge: Ubiquitous Computing und RFID in der Praxis: Visionen, Technologien, Anwendungen, Handlungsanleitungen. Springer

Guinard D, Trifa V (2009) Towards the Web of Things: Web Mashups for Embedded Devices. In : Proceedings of the 2nd Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009), Madrid, Spain, 20 April

Guinard D, Trifa V, Pham T et al. (2009) Towards physical mashups in the web of things. In: Proceedings of the 6th International Conference on Networked Sensing Systems (INSS 2009)

Hoyer V, Stanoesvka-Slabeva K, Janner T et al. (2008) Enterprise mashups: design principles towards the long tail of user needs. In: Proceedings of IEEE Services Computing, 2:601–602

Jammes F, Smit H (2005) Service-oriented paradigms in industrial automation. IEEE Transactions on Industrial Informatics, pp. 62–70

Karnouskos S, Baecker O, de Souza LMS et al. (2007) Integration of soa-ready networked embedded devices in enterprise systems via a cross-layered web service infrastructure. In: Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation (ETFA), pp. 293–300

Kennedy P, Bapat V, Kurchina P (2008) In pursuit of the perfect plant. Evolved Technologist, ISBN 978-0978921866

Liu X, Hui Y, Sun W et al. (2007) Towards service composition based on mashup. In: Proceedings of IEEE Service Computing, pp. 332–339

Luckenbach T, Gober P, Arbanowski S et al. (2005) TinyREST–a protocol for integrating sensor networks into the internet. In: Proceedings of REALWSN

Mahnke W, Leitner SH, Damm M (2009) OPC Unified Architecture. Springer, 1st edition, ISBN 3540688986

Marin-Perianu M, Meratnia N, Havinga P et al. (2007) Decentralized enterprise systems: a multiplatform wireless sensor network approach. IEEE, Wireless Communications. ISSN 1536-1284

Pidd M, Robinson S (2007) Organising insights into simulation practice. In: Proceedings of the 2007 Winter Simulation Conference, pp. 771–775

Priyantha NB, Kansal A, Goraczko M et al. (2008) Tiny web services: design and implementation of interoperable and evolvable sensor networks. In: Proceedings of the 6th ACM conference on Embedded Network Sensor Systems, Raleigh, NC, USA, pp. 253–266

SAP (2008) Towards a European Strategy for the Future Internet, http://www.europeansoftware.org/documents/SAP\s\do5(W)P\s\do5(F)utureInternet.pdf

Trifa V, Wieland S, Guinard D et al. (2009) Design and implementation of a gateway for web-based interaction and management of embedded devices In: Proceedings of the 2nd International Workshop on Sensor Network Engineering (IWSNE'09), Marina del Rey, CA, USA

Wilde E (2007) Putting things to REST. Technical Report UCB iSchool Report 2007-015, School of Information, November, UC Berkeley