SOCRADES: A Web Service based Shop Floor Integration Infrastructure

Luciana Moreira Sá de Souza, Patrik Spiess, Dominique Guinard,
Moritz Köhler, Stamatis Karnouskos, and Domnic Savio

SAP Research
Vincenz-Priessnitz-Strasse 1, D-76131, Karlsruhe, Germany
Kreuzplatz 20, CH-8008, Zurich, Switzerland
{luciana.moreira.sa.de.souza, patrik.spiess, dominique.guinard,
mo.koehler, stamatis.karnouskos, domnic.savio} @sap.com

Abstract. On the one hand, enterprises manufacturing any kinds of goods require agile production technology to be able to fully accommodate their customers' demand for flexibility. On the other hand, Smart Objects, such as networked intelligent machines or tagged raw materials, exhibit ever increasing capabilities, up to the point where they offer their smart behaviour as web services. The two trends towards higher flexibility and more capable objects will lead to a service-oriented infrastructure where complex processes will span over all types of systems — from the backend enterprise system down to the Smart Objects. To fully support this, we present SOCRADES, an integration architecture that can serve the requirements of future manufacturing. SOCRADES provides generic components upon which sophisticated production processes can be modelled. In this paper we in particular give a list of requirements, the design, and the reference implementation of that integration architecture.

1 Introduction

In the manufacturing domain, constant improvements and innovation in the business processes are key factors in order to keep enterprises competitive in the market. Manufacturing businesses are standing on the brink of a new era, one that will considerably transform the way business processes are handled.

With the introduction of ubiquitous computing on the shop floor[1], an entirely new dynamic network of networked devices can be created - an Internet of Things (IoT) for manufacturing. The Internet of Things is a concept which first appeared shortly after 2000. Until now, several approaches to describe the IoT have been undertaken of which most have focused on RFID technologies and their application ([5, 13]).

Only recently, new technologies such as Smart Embedded Devices and Sensor Networks have entered the scene and can be considered as part of the IoT [11].

[1] In manufacturing, the shop floor is the location where machines are located and products produced.

Smart Embedded Devices are embedded electronic systems which can sense their internal state and are able to communicate it through data networks. In contrast to this, Sensor Networks not only can measure internal states of their nodes, but also external states of the environment. We group these three technologies - RFID, Smart Embedded Devices, and Sensor Networks - under the notion *Smart Objects*.

Smart Objects are the nerve cells, which are interconnected through the Internet and thus build the IoT. RFID has already been proved to open fundamentally new ways of executing business processes, and the technology has already been adopted by several key players in the industry. Therefore the focus of this paper is on Smart Embedded Devices and Sensor Networks and their effects on automatic business process execution.

Although client-server architectures still play an important role in the field of business software systems, the Service Oriented Architecture (SOA) is on the move and it is foreseeable that this architectural paradigm will be predominant in the future. The integration of devices into the business IT-landscape through SOA is a promising approach to connect physical objects and to make them available to IT-systems. This can be achieved by running instances of web services on these devices, which moves the integration of back end applications, such as Enterprise Resource Planning (ERP) systems, with the devices one step forward, enabling them to interact and create an Internet of Services that collaborates and empowers the future service-based factory.

Enabling efficient collaboration between device-level SOA and services and applications that constitute the enterprise back-end on the other hand, is a challenging task. The introduction of web service concepts at a level as low as the production device or facility automation makes this integration significantly less complex. But there are still differences between device-level SOA and the one that is used in the back end. To name but a few of them, device-level services are of higher granularity, exhibit a lower reliability (especially if they are connected wirelessly) and higher dynamicity and are more focused on technical issues than on business aspects.

These differences can be overcome by introducing a middleware between the back end applications and the services that are offered by devices, service mediators, and gateways. This middleware adds the required reliability, provides means to deal with services appearing and disappearing, and allows intermediate service composition to raise the technical interfaces of low-level services to business-relevant ones.

In this paper we present the SOCRADES middleware for business integration; an architecture focused on coupling web service enabled devices with enterprise applications such as ERP Systems. Our approach combines existing technologies and proposes new concepts for the management of services running on the devices.

This paper is organized as follows: in section 2 we discuss the current state of the art in coupling technologies for shop floor and enterprise applications. Section 3 presents the requirements for our approach, followed by section 4 where we

discuss our approach. We propose a prototype for evaluating our approach in section 5 and perform an analysis in section 6. Section 7 concludes this paper.

2 Related Work

Manufacturing companies need agile production systems that can support reconfigurability and flexibility to economically manufacture products. These systems must be able to inform resource planning systems like SAP ERP in advance, about the upcoming breakdown of a whole production processes or parts of them, so that adaptation in the workflow can be elaborated.

Currently Manufacturing Execution Systems (MES) are bridging the gap between the shop floor and ERP systems that run in the back end. The International Systems and Automation Society - 95 (ISA-95) derivative from the Instrumentation Systems and Automation Society define the standards for this interface [1]. Although MES systems exist as gateways between the enterprise world and the shop floor, they have to be tailored to the individual group of devices and protocols that exist on this shop floor.

By integrating web services on the shop floor, devices have the possibility of interacting seamlessly with the back end system ([9, 8]). Currently products like SIMATIC WinCC Smart Access [2] from Siemens Automation use SOAP for accessing tag based data from devices like display panels to PC's. However, they neither provide mechanisms to discover other web-service enabled devices, nor mechanisms for maintaining a catalogue of discovered devices.

The domain of Holonic Manufacturing Execution Systems (HMS) [6] is also relevant to our work. HMS are used in the context of collaborative computing, and use web service concepts to integrate different sources and destinations inside a production environment. They do, however, not offer support to process orchestration or service composition.

Amongst others, European Commission funded projects like SIRENA [4] showed the feasibility and benefit of embedding web services in production devices. However, since these were only initial efforts for proving the concept, not much attention has been given to issues such as device supervision, device life cycle management, or catalogues for maintaining the status of discovered devices, etc. The consortium of the SOCRADES project has integrated partners, code and concepts from SIRENA, and aims to further design and implement a more sophisticated infrastructure of web-service enabled devices. SODA (www.soda-itea.org) aims at creating a comprehensive, scalable, easy to deploy ecosystem built on top of the foundations laid by the SIRENA project.

The SODA ecosystem will comprise a comprehensive tool suite and will target industry standard platforms supported by wired and wireless communications. Although EU projects like SIRENA showed the feasibility and benefit of embedding web services in devices used for production, they do not offer an infrastructure or a framework for device supervision or device life cycle. They neither do provide a catalogue for maintaining the status of discovered devices [4]. Changes due to the current development are moving towards a more promis-

ing approach of integrating shop floor devices and ERP systems more strongly [14].

Some authors are criticizing the use of RPC-style interaction in ubiquitous computing [12] (we consider the smart manufacturing devices a special case of that). We believe this does not concern our approach, since web services also allow for interaction with asynchronous, one-way messages and publish-subscribe communication.

SAP xApp Manufacturing Integration and Intelligence (SAP xMII) is a manufacturing intelligence portal that uses a web server to extract data from multiple sources, aggregate it at the server, transform it into business context and personalize the delivered results to the users [7]. The user community can include existing personal computers running internet browsers, wireless PDAs or other UIs. Using database connectivity, any legacy device can expose itself to the enterprise systems using this technology.

The drawback of this product is that every device has to communicate to the system using a driver that is tailored to the database connectivity. In this way, SAP xMII limits itself to devices or gateway solutions that support database connectivity.

In [10], we proposed a service-oriented architecture to bridge between shop floor devices and enterprise applications. In this paper however, building on both our previous work and SAP xMII, we show how the already available functionality of xMII can be leveraged and extended to provide an even richer integration platform. The added functionality comprises integration of web service enabled devices, making them accessible through xMII, and supporting the software life cycle of embedded services. This enables real-world devices to seamlessly participate in business processes that span over several systems from the back end through the middleware right down to the Smart Objects.

3 System Requirements

As embedded technology advances, more functionality that currently is hosted on powerful back end systems and intermediate supervisory devices can now be pushed down to the shop floor level. Although this functionality can be transferred to devices that have only a fraction of the capabilities of more complex systems, their distributed orchestration in conjunction with the fact that they execute very task-specific processing, allows us to realise approaches that can outperform centralised systems in means of functionality. By embedding web services on devices, these can become part of a modern Enterprise SOA communication infrastructure.

The first step to come closer to realize this vision, is to create a list of requirements. We have come up with this list through interviews with project partners and customers from the application domain, as well as a series of technical workshops with partners form the solution domain. As usually done in software engineering, we separated the list into functional and non-functional requirements.

Functional Requirements

- **WS based direct access to devices:** Back end services must be able to discover and directly communicate with devices, and consume the services they offer. This implies the capability of event notifications from the device side, to which other services can subscribe to.
- **WS based direct access to back end services:** Most efforts in the research domain today focus on how to open the shop floor functionality to the back end systems. The next challenge is to open back end systems to the shop floor. E.g. devices must be able to subscribe to events and use enterprise services. Having achieved that, business logic executing locally on shop floor devices can now take decisions not only based on its local information, but also on information from back end systems.
- **Service Discovery:** Having the services on devices will not be of much use if they can not be dynamically discovered by other entities. Automatic service discovery will allow us to access them in a dynamic way without having explicit task knowledge and the need of a priori binding. The last would also prevent the system from scaling and we could not create abstract business process models.
- **Brokered access to events:** Events are a fundamental pillar of a service based infrastructure. Therefore access to these has to be eased. As many devices are expected to be mobile, and their online status often change (including the services they host), buffered service invocation should be in-place to guarantee that any started process will continue when the device becomes available again. Also, since not all applications expose web services, a pull point should be realised that will offer access to infrastructure events by polling.
- **Service life cycle management:** In future factories, various services are expected to be installed, updated, deleted, started, and stopped. Therefore, we need an open ways of managing their life cycle. Therefore the requirement is to provide basic support in the infrastructure itself that can offer an open way of handling these issues.
- **Legacy device integration:** Devices of older generations should be also part of the new infrastructure. Although their role will be mostly providing (and not consuming) information, we have to make sure that this information can be acquired and transformed to fit in the new WS-enabled factory. Therefore the requirement is to implement gateways and service mediators to allow integration of the non-ws enabled devices.
- **Middleware historian:** In an information-rich future factory, logging of data, events, and the history of devices is needed. The middleware historian is needed which offers information to middleware services, especially when an analysis of up-to-now behavior of devices and services is needed.
- **Middleware device management:** Web service enabled devices, will contain both, static and dynamic data. This data can now be better and more reliably integrated to back end systems offering a more accurate view of the shop floor state. Furthermore by checking device data and enterprise inventory, incompatibilities can be discovered and tackled. Therefore we require

approaches that will effectively enable the full integration of device data and their exploitation above the device-layer.

Non-Functional Requirements

- **Security support:** Shop floors are more or less closed environments with limited and controlled communication among their components. However, because of open (and partially wireless) communication networks, this is fundamentally changing. Issues like confidentiality, integrity, availability must be tackled. In a web service mash-up - as the future factory is expected to be -, devices must be able to a) authenticate themselves to external services and b) authenticate/control access to services they offer.
- **Semantics support:** This requirement facilitates the basic blocks primarily for service composition but also for meaningful data understanding and integration. Support for the usage of ontologies and semantic-web concepts will also enhance collaboration as a formal description of concepts, terms, and relationships within a manufacturing knowledge domain.
- **Service composition:** In a SOA infrastructure, service composition will allow us to build more sophisticated services on top of generic ones, therefore allowing thin add-ons for enhanced functionality. This implies a mixed environment where one could compose services a) at device level b) at back end level and c) in a bidirectional cross-level way.

In the above list we have described both, functional and non-functional requirements. In our architecture these requirements will be realized through components, each one offering a unique functionality.

4 Architecture

4.1 Overview

In this chapter, we present a concrete integration architecture focusing on leveraging the benefits of existing technologies and taking them to a next level of integration through the use of DPWS and the SOCRADES middleware. The architecture proposed in Figure 1 is composed of four main layers: Device Layer, SOCRADES middleware (consisting of an application and a device services part), xMII, and Enterprise Applications.

The Device Layer comprises the devices in the shop floor. These devices when enabled with DPWS connect to the SOCRADES middleware for more advanced features. Nevertheless, since they support web services, they provide the means for a direct connection to Enterprise Applications. For the intermediate part of the SOCRADES architecture, bridging between enterprise and device layer, we identified an SAP product that partly covered our requirements: SAP xApp Manufacturing Integration and Intelligence (SAP xMII). The features already available in xMII are:

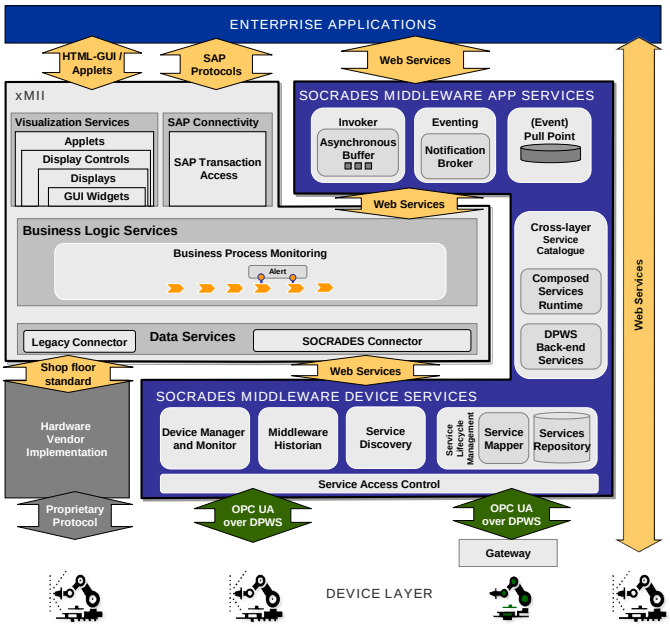

Fig. 1. SOCRADES Integrated Architecture

- Connectivity to non web service enabled devices via various shop floor communication standards
- Graphical modelling and execution of business rules
- Visualization Services
- Connectivity to older SAP software through SAP-specific protocols

We decided not to re-implement that functionality but use it as a basis and extend it by what we call the SOCRADES middleware. The SOCRADES middleware and xMII perform together a full integration of devices with ERP systems, adding functionalities such as graphical visualization of device data and life cycle management of services running on the devices. In this setting, xMII provides the handling of business logic, process monitoring and visualization of the current status of the devices.

Finally, the connection with Enterprise Applications is realized in three ways. SAP xMII can be used to generate rich web content that can be integrated into the GUI of an enterprise system in mash-up style. Alternatively, it can be used to establish the connection to older SAP systems using SAP-specific protocols.

Current, web service based enterprise software can access devices either via web services of the SOCRADES middleware, benefiting from the additional functionality, or they can directly bind against the web services of DPWS-enabled devices. The data delivered to Enterprise Applications is currently provided by xMII. Nevertheless with the introduction of the SOCRADES middleware and

the use of DPWS, this data can be also delivered directly by the regarding devices, leaving to xMII only the task of delivering processed data that requires a global view of the shop floor and of the business process.

4.2 Features and Components of the SOCRADES Middleware

The SOCRADES middleware is the bridging technology that enables the use of features of existing software systems with DPWS enabled devices. Together with SAP xMII, this middleware connects the shop floor to the top floor, providing additional functionality not available in either one of these layers. Although direct access from an ERP system to devices is possible, the SOCRADES middleware simplifies the management of the shop floor devices. In the following, we list this additional functionality and show how the components of the architecture implement them.

Brokered Access to Devices. Brokered access means to have an intermediate party in the communication between web service clients and servers that adds functionality. Example are asynchronous invocations, a pull point for handling events, and a publish-subscribe mechanism for events. Asynchronous invocations are useful when dealing with devices that are occasionally connected so that invocations have to be buffered until the device re-appears; they are implemented by the Invoker component. Pull points enable applications to access events without having to expose a web service interface to receive them. The application can instruct the pull point to buffer events and can obtain them by a web service call whenever it is ready. Alternatively, to be notified immediately, the application can expose a web service endpoint and register it at the notification broker for any type of event.

Service Discovery: The service discovery components carries out the actual service discovery on the shop floor level. This component is distributed and replicated at each physical site because the DPWS discovery mechanism *WS-Discovery* relies on UDP multicast, a feature that may not be enabled globally across all subsidiaries in a corporate network. All discovered devices from all physically distributed sites and all the services that each device runs are then in a central repository called *Device Manager and Monitor*, which acts as the single access point where ERP systems can find all devices even when they have no direct access to the shop floor network.

Device Supervision: Device Management and Monitor and DPWS Historian provide the necessary static and dynamic information about each DPWS-enabled physical device available in the system. The *device manager* holds any static device data of all on-line and off-line devices while the *device monitor* contains information about the current state of each device. The middleware historian can be configured to log any event occurring at middleware level for later diagnosis

and analysis. Many low-level production systems feature historians, but they are concerned with logging low-level data that might be irrelevant for business-level analysis. Only a middleware historian can capture high-level events that are constructed within this architectural layer.

Service Life Cycle Management: Some hardware platforms allow exchanging the embedded software running on them via the network. In a service-enabled shop floor this means that one can update services running on devices. The management of these installed services is handled through the use of the Service Mapper and Services Repository. These components together make a selection of the software that should run in each device and perform the deployment.

Cross-Layer Service Catalogue: The cross-layer service catalogue comprises two components. One is the Composed Services Runtime that executes service composition descriptions, therefore realizing service composition at the middleware layer. The second component is the DPWS device for back end services that allows DPWS devices to discover and use a relevant set of services of the ERP system.

The Composed Services Runtime is used to enrich the services offered by the shop floor devices with business context, such as associating an ID read from an RFID tag with the corresponding order. A compound service can deliver this data by both invoking a service on the RFID reader, and from a warehouse application. A Composed Services Runtime, which is an execution engine for such service composition descriptions, e.g., BPEL [3], is placed in the middleware because only from there, all DPWS services on the shop floor as well as all back end services can be reached.

Another requirement is that shop floor devices must be able to access enterprise application services, which can be achieved by making a relevant subset available through the DPWS discovery. This way, devices that run DPWS clients can invoke back end services in exactly the same way they invoke services on their peer devices. Providing only the relevant back end services allows for some access control and reduces overhead during discovery of devices. Co-locating both sub-components in the same component has the advantage that also the composed services that the Composed Services Runtime provides, can be made available to the devices through the virtual DPWS device for back end services.

Security support: The (optional) security features supported by the middleware are role-based access control of devices communication to middleware and back end services and vice versa. Event filtering based on roles is also possible. Both the devices as well as back end and middleware services have to be authorized when they want to communicate. Access control is enforced by the respective component. Additionally, message integrity and confidentiality is provided by the WS-Security standard.

To demonstrate the feasibility of our approach and to make some first evaluations, we implemented a simple manufacturing scenario. We used a first implementation of our architecture to connect two DPWS-enabled real-world devices with an enterprise application.

5 Reference Implementation

In order to prove the feasibility of our concept, we have started realising a reference implementation. From a functional point of view, it demonstrates two of the most important incentives for the use of standardized device level web services in manufacturing: flexibility and integration with enterprise software. Indeed, the scenario shows DPWS-enabled devices can be combined easily to create higher-level services and behaviours that can then be integrated into top-floor applications.

The business benefits from adopting such an architecture are numerous:

- lower cost of information delivery
- increased flexibility and thus total cost of ownership (TCO) of machines.
- increased visibility of the entire manufacturing process to the shop floor.
- ability to model at the enterprise layer processes with only abstract view of the underlying layer, therefore easing the creation of new applications and services from non-domain experts.

5.1 Scenario

To support this idea we consider a simple setting with two DPWS devices:

- A robotic arm that can be operated through web service calls. Additionally it offers status information to subscribers through the SOCRADES eventing system.
- A wireless sensor node providing various information about the current environment, delivered as events. Furthermore, the sensor nodes provide actuators that are accessible through standard service calls.

The manufacturing process is created on the shop floor using a simple service composition scheme: from the atomic services offered by the arm (such as start/stop, etc.) a simple manufacturing process p is created. The robot manipulates heat-sensitive chemicals. As a consequence it is identified that the manufacturing process cannot continue if the temperature rises above 45 ℃.

The robot may not have a temperature sensor (or this is malfunctioning), but as mentioned before the manufacturing plant is equipped with a network of wireless sensor nodes providing information about the environment. Thus, in order to enforce the business rule, the chief operator uses a visual composition language to combine p with the temperature information published by the service-enabled sensor node: t.

In pseudo code, such a rule looks like:

```
if (t > 45) then p.stopTransportProcess();
```

Furthermore, the operator instantiates a simple gauge fed with the temperature data (provided by t). For this purpose he uses a manufacturing intelligence software and displays the gauge on a screen situated close the robot.

Finally, the sales manager can also leverage the service oriented architecture of this factory. Indeed, 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.

5.2 Components

This section describes the architecture of our prototype from an abstract point of view. Its aim is to understand the functionality whose concrete implementation will be described within the next section.

Functional Components The system comprises four main components as shown on Figure 2 that we shall briefly describe:

- **Smart Devices:** Manufacturing devices, sensors and Smart Things (i.e. Smart Objects) are the actors forming an Internet of Services in the factory as well as outside of the factory. They all offer web service interfaces, either directly or through the use of gateways or service mediators. Through these interfaces they offer functional services (e.g. start/stop, swap to manual/automatic mode) or status information (e.g. power consumption, mode of operation, usage statistics, etc.).
- **Composed Service:** The component aggregates the services offered by smart objects. Indeed, it is in charge of exposing coarse-grained services to the upper layers. In the case of the robotic arm for instance, it will consume the `open()`, `close()` and `move(...)`, methods and use them to offer a `doTransportProcess (...)` service.
- **Business Logic Services and Visualisation Services:** In our prototype, the business logic services are supported by a service composition engine and visualized using a visualization toolkit. The former component is used to model business rules or higher-level processes, known as business logic services in our architecture. As an example the operator can use it to create the business rules exposed above. The latter component is used to build a plant-floor visualisation of the devices' status and the overall process execution. As an example the operator can instantiate and use a set of widgets such as gauges and graphs to monitor the status of the machines. The production manager can also use it to obtain real-time graphs of the process execution and status.

– **Enterprise Applications:** This is the place of high-end business software such as ERPs or PLMs. 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 machines failures and plant-floor information on the process visualization and work-flow. Furthermore, business actions (e.g. inform customers about a possible delay) can be executed based on this information.

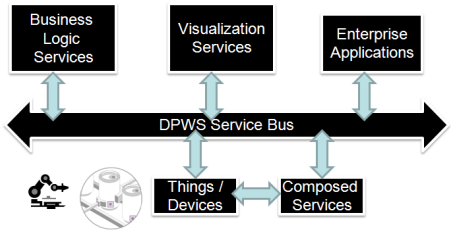

Fig. 2. The DPWS service bus.

Cross-Component Communication In a mash-up, the architecture is not layered but rather flat, enabling any functional component to talk to any other. Such architectures need a common denominator in order for the components to be able to invoke services on one another. In our case the common denominator is the enhanced DPWS we developed. Each component is DPWS-enabled and thus, consumes DPWS services and exposes a DPWS interface to invoke the operations it offers. The service invocations can be done either synchronously or asynchronously via the web service eventing system. For instance the temperature is gathered via a subscription to the temperature service (asynchronous) whereas the transport process is stopped by invoking an operation on the process middleware. Figure 2 depicts the architecture by representing the components connected to a common (DPWS) ESB (Enterprise Service Bus).

5.3 Implementation

The system described in this paper is a reference implementation of concepts described in the architecture rather than a stand-alone concept. Therefore it uses and extends several software and hardware components rather than writing them from scratch. In this section we will briefly describe what these components are and how they interact together, taking a bottom up approach.

Functional Components

- **Smart Devices:** The wireless sensor network providing temperature information is implemented using the Sun Microsystems' SunSPOT sensor nodes. Since the nodes are not web services enabled, we had to implement a gateway (as described in our architecture), that would capture the temperature readings and provide it via DPWS as services one can subscribe to. The gateway component hides the communication protocol between the SunSPOTs and exposes their functionalities as device level web services (DPWS). More concretely the SunSPOT offer services for sensing the environment (e.g. `getTemperature()`) or providing output directly on the nodes (e.g. `turnLightOn(Color)`). The robotic arm was implemented as a clamp offering DPWS services for both monitoring and control. The clamp makes these operations available as DPWS SOAP calls on a PLC (Programmable Logic Controller) over gateway. For monitoring services (e.g. `getPowerConsumption()`) the calls are issued directly on the gateway standing for the clamp. For control services the idea is slightly different.
- **Composed Service:** Typical operations at the clamp level are `openClamp()` and `closeClamp()`. In order to consistently use these operations on the top-floor we need to add some business semantics already on the shop floor. This is the role of composed services which aggregate an number of coarse-grained operations (e.g. `openClamp()`) and turn them into higher level services. This way the `start()`, `openClamp()`, `closeClamp()`, `move(x)`, `stop()` operations are combined to offer the `startTransportProcess()` service.
- **Business Logic Services and Vizualisation Services:** Services offered by both the sensors and the clamp are combined to create a business rule. The creation of this business logic service is supported by xMII, SAP's Manufacturing Integration and Intelligence software. As mentioned before, the aim of this software is firstly to offer a mean for gathering monitoring data from different device aggregators on the shop floor such as MESs (Manufacturing Execution Systems). This functionality is depicted on Figure 3.

  Since the SOCRADES infrastructure proposes to DPWS-enable all the devices on the plant-floor, we can enhance the model by directly connecting the devices to xMII. Additionally, xMII offers a business intelligence tool. Using its data visualization services we create a visualization of process-related and monitoring data. Finally, we use the visual composition tool offered by xMII to create the rule. Whenever this rule is triggered the `stopTransportProcess()`operation is invoked on the middleware to stop the clamp.
- **Enterprise Applications:** Whenever the business rule is triggered, xMII invokes the `updateOrderStatus()`on the ERP. As mentioned before this latter component displays the failure and its consequences (i.e. a delay in the production) in the orders' list. Additionally, if the alert lasts for a while, it informs the customer by email providing him with information about a probable delay.

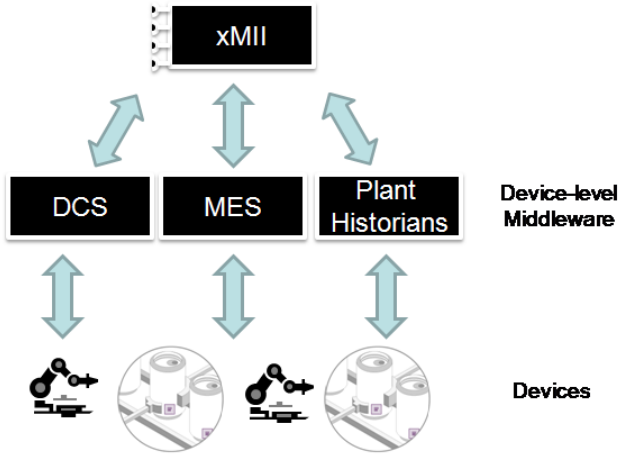

Fig. 3. xMII indirect device connectivity.

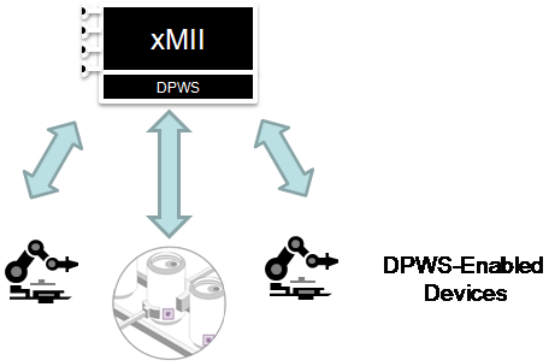

Fig. 4. Direct connectivity to the DPWS devices.

Cross-Component Communication Figure 5 presents the communication amongst the components whenever the business rule is triggered. At first the SunSPOT dispatches the temperature change by placing a SOAP message on the DPWS service bus. The xMII is subscribed to this event and thus, receives the message and feeds it to its rules engine. Since the reported temperature is above the threshold xMII fires the rule. As a consequence it invokes the `stopTransportProcess()`operation on the Process Service middleware. This component contacts the clamp and stops it. Furthermore, xMII triggers the `updateOrderStatus()`operation on the ERP. This latter system update the status of the concerned order accordingly and decides whether to contact the customer to inform him by email about the delay.

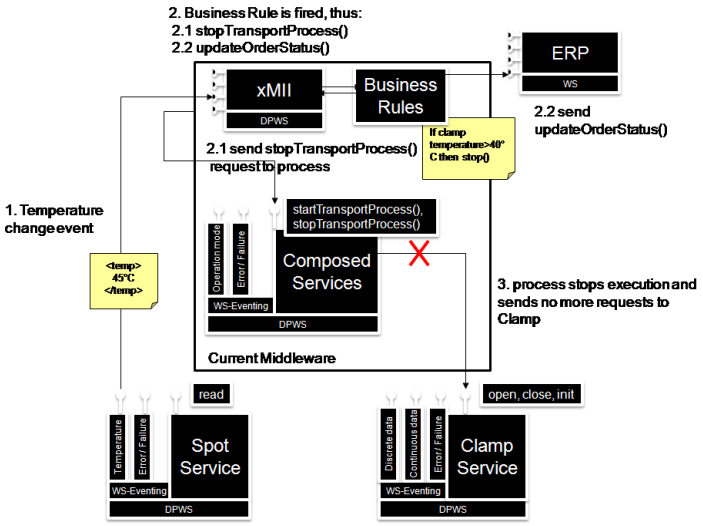

Fig. 5. Interactions when the business rule is triggered.

6 System Analysis

In this section we will discuss the properties of our architecture and give decision makers a framework at hand through which they can assess the concrete value of our system for their organisation. Since the work we are presenting in this paper is part of ongoing research, we think it is helpful to have such a framework, in particular to assess future work.

In the field of Systems Management several standards exist [ref. Standards, ITIL, etc.] which aim to support a structured dealing with IT systems. One framework in particular helpful for central corporate functions such as produc-

tion is the ISO model FCAPS (Fault, Configuration, Administration, Performance, Security). Although being a framework for network management, it is relevant for our architecture because it is enabling low level networked interaction between Smart Objects. Here we will give a first attempt to evaluate the architecture.

- **Fault Management:** Since our system will be part of the manufacturing IT-landscape we need to manage both, faults of particular parts of the manufacturing process and faults in our system. Due to the tight integration these types of faults inherently become the same. In particular the SOA based approach of device integration enables the user to identify faults in his production process, at a level never seen before. It also gives the possibility to build redundancy at system critical stages which ensures fast recovery from local failures. Finally the flexibility given by our SOA approach lets the user decide to what extend he wants to introduce capabilities of quick fault recovery, depending on his individual needs.
- **Configuration Management:** Mainly the two components *Service Lifecycle Management* and *Cross-Layer Service Catalogue* support dynamic configuration management. However, at the current point of view we see code updated to Smart Devices as a major challenge which until today has not been resolved sufficiently. Configuration also includes the composition of services into higher-level services. In a future version, our Service Discovery module will use semantic annotation of services to find appropriate service instances for online service composition. Using ontologies to specify the behaviour and parameters of web services in their interface descriptions and metadata allows flexible service composition. Especially in the very well defined domain of manufacturing we can make use of existing ontologies that describe production processes.
- **Administrative Management:** The *Device Manager* provides the necessary static and dynamic information about each Smart Device. Through the strict use of web-service interfaces, it will be possible to easily integrate devices into management dash-boards. Through this technically we allow easy and user friendly access to Smart Devices. However, taking the possibly very large number of devices into account, we belief that our middle-ware has deficiencies in offering this user friendly administration. Although this problem is subject to other fields of research such as sensor networks, (e.g, macro programming), we will dedicate our research efforts to the problem.
- **Performance Management:** Already now we can say that local components of our system will scale well in regards to total amount of Smart Objects and their level of interaction. This can be justified since all interaction occurs locally and only a limited amount of Smart Objects is needed to fulfil a particular task. However, it is still an open question, if our system will scale well on a global scale and to what extend it will need to be modularized. For example we will need to investigate whether central components such as device and service registries should operate on a plant level or on

a corporate level, which could mean that these parts would have to handle several millions or even billions of devices at the same time.
- **Security Management:** As mentioned in the security support section of the architecture, our system can make use of well established security features which already are part of web-service technologies and their protocols such as DPWS. It is most likely that we will have to take into account industry specific security requirements, and it will be interesting to see, if we can deliver a security specification which satisfies all manufacturing setups.

7 Conclusions

In this paper we have presented SOCRADES, a Web Service based Shop Floor Integration Infrastructure. With SOCRADES we are offering an architecture including a middleware which support connecting Smart Devices, i.e. intelligent production machines from manufacturing shop floors, to high-level back-end systems such as an ERP system. Our integration strategy is to use web services as the main connector technology. This approach is motivated by the emerging importance of Enterprise Service Oriented Architectures, which are enabled through web services.

Our work has three main contributions: First, we elaborated and structured a set of requirements for the integration problem. Second, we are proposing a concrete architecture containing of components which realized the required functionality of the system. Our third contribution is a reference implementation of the SOCRADES architecture. In this implementation we have demonstrated the full integration of two Smart Devices into and enterprise system. We showed that it is possible to connect Smart Devices to an ERP system, and describe how this is done.

Our next steps include integrating a prototype in a bigger setup and testing it with live production systems.

8 Acknowledgments

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

1. Instrumentation Systems and Automation Society. http://www.isa.org/.
2. SIMATIC WinCC flexible. http://www.siemens.com/simatic-wincc-flexible/.
3. Web Services Business Process Execution Language Version 2.0 (OASIS Standard), April 2007. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html.

4. H. Bohn, A. Bobek, and F. Golatowski. SIRENA - Service Infrastructure for Real-time Embedded Networked Devices: A service oriented framework for different domains. In *International Conference on Systems and International Conference on Mobile Communications and Learning Technologies (ICNICONSMCL'06)*, page 43, Washington, DC, USA, 2006. IEEE Computer Society.
5. E. Fleisch and F. Mattern, editors. *Das Internet der Dinge: Ubiquitous Computing und RFID in der Praxis:Visionen, Technologien, Anwendungen, Handlungsanleitungen*. Springer, 2005.
6. L. Gaxiola, M. de J. Ramírez, G. Jimenez, and A. Molina. Proposal of Holonic Manufacturing Execution Systems Based on Web Service Technologies for Mexican SMEs. In *HoloMAS*, pages 156–166, 2003.
7. G.Gorbach. Pursuing manufacturing excellence through Real-Time performance management and continuous improvement. ARC Whitepaper, April 2006.
8. F. Jammes, A. Mensch, and H. Smit. Service-Oriented Device Communications using the Devices Profile for Web Services. In *MPAC '05: Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing*, pages 1–8, New York, NY, USA, 2005. ACM Press.
9. F. Jammes and H. Smit. Service-oriented paradigms in industrial automation. *IEEE Transactions on Industrial Informatics*, 1:62–70, 2005.
10. S. Karnouskos, O. Baecker, L. M. S. de Souza, and P. Spiess. Integration of SOA-ready Networked Embedded Devices in Enterprise Systems via a Cross-Layered Web Service Infrastructure. In *12th IEEE Conference on Emerging Technologies and Factory Automation*, 2007.
11. A. Reinhardt. A Machine-To-Machine "Internet Of Things". *Business Week*, April 2004.
12. U. Saif and D. J. Greaves. Communication Primitives for Ubiquitous Systems or RPC Considered Harmful. In *21st International Conference of Distributed Computing Systems (Workshop on Smart Appliances and Wearable Computing)*, Los Alamitos, CA, USA, 2001. IEEE Computer Society.
13. C. R. Schoenberger. RFID: The Internet of Things. *Forbes*, (18), March 2002.
14. E. Zeeb, A. Bobek, H. Bohn, and F. Golatowski. Service-Oriented Architectures for Embedded Systems Using Devices Profile for Web Services. In *21st International Conference on Advanced Information Networking and Applications Workshops.*, 2007.