Chapter 4
Promising Technologies for SOA-based Industrial Automation Systems

François Jammes, Stamatis Karnouskos, Bernard Bony, Philippe Nappey, Armando W. Colombo, Jerker Delsing, Jens Eliasson, Rumen Kyusakov, Petr Stluka, Marcel Tilly, Thomas Bangemann

Abstract In the last years Service-Oriented Architectures have been extensively used in order to enable seamless interaction and integration among the various heterogeneous systems and devices found in modern factories. The emerging Industrial Automation Systems are increasingly utilizing them. In the cloud-based vision of IMC-AESOP such technologies take an even more key role as they empower the backbone of the new concepts and approaches under development. Here we report about the investigations and assessments performed to find answers for some of the major questions that arise as key when technologies have to be selected and used in an industrial context utilizing Service-Oriented Architecture (SOA) based distributed large scale Process Monitoring and Control system. Aspects of integration, real-timeness, distributeness, event-based interaction, service-enablement etc. are approached from different angles and some of the promising technologies are analysed and assessed.

François Jammes, Bernard Bony, Philippe Nappey
Schneider Electric, France e-mail: francois2.jammes@schneider-electric.com,bernard.bony@schneider-electric.com,philippe.nappey@schneider-electric.com

Stamatis Karnouskos
SAP, Germany e-mail: stamatis.karnouskos@sap.com

Armando W. Colombo
Schneider Electric & University of Applied Sciences Emden/Leer, Germany e-mail: armando.colombo@schneider-electric.com,awcolombo@technik-emden.de

Jerker Delsing, Jens Eliasson, Rumen Kyusakov
Luleå University of Technology, Sweden, e-mail: jerker.delsing@ltu.se,jens.eliasson@ltu.se,rumen.kyusakov@ltu.se

Petr Stluka
Honeywell, Czech Republic e-mail: petr.stluka@honeywell.com

Marcel Tilly
Microsoft, Germany, e-mail: marcel.tilly@microsoft.com

Thomas Bangemann
ifak, Germany e-mail: thomas.bangemann@ifak.eu

4.1 Introduction

Current industrial process control and monitoring applications are facing many challenges as the complexity of the systems increases and the systems evolve from synchronous to asynchronous. When hundreds of thousands of devices and service-oriented systems are asynchronously interconnected and share and exchange data and information, i.e., services, for monitoring, controlling and managing the processes, key challenges such as interoperability, real-time performance constraints, among others, arise and need to be addressed.

The SOA-based approach proposed by the European R&D projects SOCRADES and subsequently IMC-AESOP [12], is addressing some of those challenges. The vision pursued is shown in Fig. 4.1; according to which the industrial process environment is mapped into a "Service Cloud", i.e. devices and applications distributed across the different layers of the enterprise are exposing their characteristics and functionalities as "services". Additionally these devices and systems are able to access and use those "services" located in the cloud [10, 13, 11].

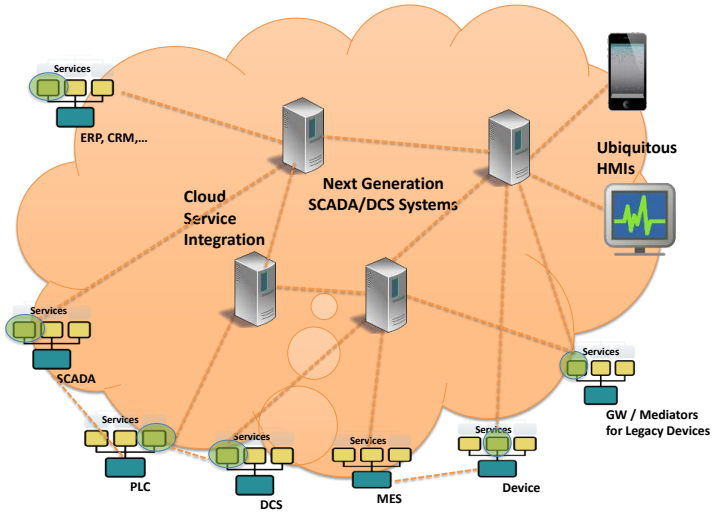

Fig. 4.1 IMC-AESOP Approach: a Distributed Dynamically Collaborative Service-oriented SCADA/DCS System

The outcomes of the first set of industry technology investigations and pilot applications carried out according to the IMC-AESOP project vision [12, 13], reveals four major challenges that may need to be addressed:

I Real-time SOA: Determine the real-time limits of bringing SOA inside the high performance control loops of process monitoring and control (e.g. is it possible to provide service-oriented solutions targeting the one millisecond performance range?)

II Large scale distributed process control and monitoring system: Is it feasible to dynamically design, deploy, configure, manage and maintain an open plant/enterprise wide system, with thousands of devices and systems operating under process real-time constraints and still comply to ISA-95 (www.isa-95.com) and PERA (www.pera-net) architectures ?

III Process Monitoring and Control Systems operating in an asynchronous mode, e.g. Distributed event based systems: Which are the technological consequences and limits of those asynchronous SCADA/DCS platforms when compared to traditional implemented periodic systems? Is it possible to integrate asynchronous and synchronous systems, e.g. for legacy system integration?

IV Service specification: Which methodology and tools are the most suitable to identify and specify the semantics for interoperable (standard / common / specific) Web services based monitoring and control (from business process to devices)?

In this work we present the results of the investigations and assessments performed [8] to find some of the answers for those four guiding questions when technologies have to be selected and used in a Service-Oriented Architecture (SOA) based Distributed Large Scale Process Control and Monitoring System. First we present a description and assessment of the most suitable technologies for addressing the four challenges described above in the area of industrial automation. Subsequently the results of the assessment synthesizing the technologies that are being used to implement the IMC-AESOP approach are shown together with highlights and some outlook for the future.

4.2 Internet Technologies for Industrial Automation

In regard of the four main challenges addressed in the introduction, several technologies have been identified as the major candidates for being used to develop such a Cloud of SCADA-/DCS-Services. Establishing an exhaustive list was not looked for (it would probably be impossible to achieve), but the major intention is to offer a compilation/screening of suitable SOA-based technologies, selected following the following main criteria:

- The technology trends reported in the most recent available publications in conferences and journals;
- The technologies that are proposed as outcomes of on-going standardisation activities;
- The potential industrial availability at short term either as open-source solutions and/or supplied by the IMC-AESOP technology-provider partners;
- The originality and innovation associated to the technology;
- The potential use of a technology by the end-user industry.

Some key technologies identified include: DPWS, EXI, CoAP, REST, OPC-UA, Distributed Service Bus, Complex Event Processing (CEP), Semantic Technologies.

4.2.1 DPWS and EXI

DPWS is recognized as a very good SOA device level protocol profile. Among all Web services protocols, it selects the most appropriate ones, such as WS-Discovery and WS-Eventing above SOAP, for implementation in constrained embedded devices. It provides capabilities such as interoperability, plug and play, and integration capability. Several projects such as SOCRADES (www.socrades.net) and SIRENA www.sirena-itea.org have demonstrated its capabilities. However, it does not provide alone real time performance in the millisecond range [6]. However, if we couple DPWS with EXI (wwww3.org/XML/EXI), this performance target is achievable.

When looking at the real time challenge, the performance that is evaluated and measured is defined as the time to send an event from one device application to another remote device application on a local network. This is done taking into account the time periods required to go through emitter and receiver stacks and to go through the local network, in a one way asynchronous event transmission.

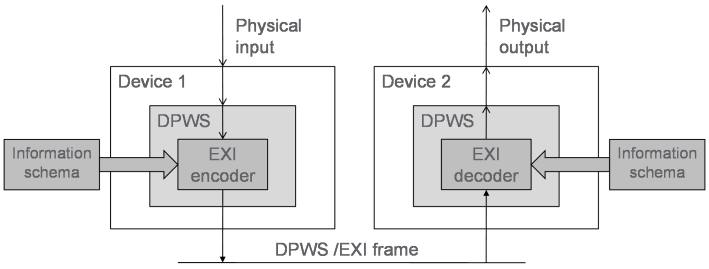

Fig. 4.2 DPWS / EXI Integration

In the example shown in the Fig. 4.2, two remote devices are connected by a physical network (e.g. Ethernet). The first device is detecting a data change on one of its physical inputs, and is sending this information through the network to the second device, which then generates a corresponding physical output. Both devices are using DPWS [4] in order to exchange the information, which provides all the customer values of DPWS (interoperability, plug and play, integration capability). They integrate inside the DPWS stack the EXI encoder or decoder capability in order to add real time performance to the standard DPWS values.

After this exchange, the first device, when receiving an input change, will translate this physical event into a DPWS / EXI network event, using the combined capabilities of the DPWS stack and of the EXI encoder, which was programmed or configured according to the information schema. The second device, when receiving the network event, will decode the frame and transform it into an output change.

4.2.2 EXIP – EXI Project

The implementation of XML/EXI technology provides a very generic framework for describing, implementing, and maintaining complex systems and interactions. However, the usage of XML, even when used with a binary compressed representation, can result in a too high overhead for deeply constrained devices. Furthermore, the application of complex schemas and WSDL descriptions can make versioning difficult since the XML/EXI parsers might require updated grammars for optimal performance.

In some cases, the use of a more simple data representation such as JSON and SenML might be sufficient, especially for very low-cost sensors and actuators. However, the implementation of different data representation techniques between the resource constrained devices and more capable systems requires service gateways that convert these data formats. Using service gateways and mediators introduces complexity in the provisioning and maintenance of the systems. In such scenario, it is beneficial to use EXI all the way down to the sensor and actuator devices.

Although the EXI format is designed for high compression and fast processing, its deployment on deeply constrained devices such as wireless sensor nodes is challenging due to RAM and programming memory requirements. The EXIP open source project [14] is providing efficient EXI processing for such embedded devices. The EXIP prototype implementation is specially designed to handle typed data and small EXI messages efficiently as this is often required in process monitoring and control applications for sensor data acquisition.

The EXIP project also includes a novel EXI grammar generator that efficiently converts an EXI encoded XML Schema document into EXI grammar definitions. These grammars are then used for schema-enabled processing which provides a better performance than schema-less mode. This grammar generator enables the use of dynamic schema-enabled processing in constrained environments as the EXI encoded XML Schemas are much lighter to transmit and process. The use of EXI representation of the schemas is possible because the XML Schema documents are plain XML documents and as such they have analogous EXI representation.

Working with the EXI representation of the XML Schema definitions brings all the performance benefits of the EXI itself i.e. faster processing and more compact representation. The use of different XML schemas and even different version of these schemas at run time is challenging. For that reason, an important future work investigation is the support for XML Schema evolutions in the SOA implementations.

Another important aspect is the definition of EXI profile for implementation in industrial environments that will guarantee interoperability and optimal performance of the EXI processing. This profile must specify what options should be used in the EXI headers and how the schema information is communicated between the devices and systems.

The main results of the performed evaluation of EXI show that:

- The use of EXI provides significant reduction of the exchanged mes-sage sizes. Compression ratios up to 20-fold may be obtained for some types of messages. Although the experiment has been performed on a high-speed wired Ethernet network, it is also expected that low-bandwidth networks such as those found in wireless sensor networks would also strongly benefit from the use of EXI.
- Performance improvements are less significant: only an improvement by a factor of 2 has been measured. This is due in part to the inherent complexity of EXI, which is computation-intensive, but also to the overhead of the underlying message exchange protocols (HTTP and SOAP in the experiment). Further experiments using more efficient protocols, such as a simple TCP protocol or the new CoAP protocol, could demonstrate that EXI is also relevant for high-performances applications.

4.2.3 CoAP

In the era of lightweight integration especially of resource-constraint devices with web technologies, a new application protocol is proposed within the Internet Engineering Task Force (IETF) i.e. the Constrained Application Protocol (CoAP) [2, 21]. CoAP provides a method/response interaction model between application end-points, supports built-in resource discovery, and includes key web concepts such as URIs and content-types. CoAP also easily translates to HTTP for seamless integration with the web, while meeting specialized requirements such as event-based communication, multicast support, very low overhead and simplicity for constrained environments.

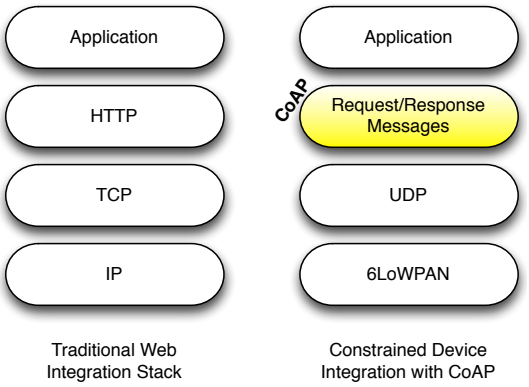

Fig. 4.3 CoAP lightweight integration vs. the heavy HTTP integration

As depicted in Fig. 4.3, CoAP relies on UDP instead of TCP that is used by default for HTTP integration. UDP provides advantages for low overhead and multicast support. CoAP is REST centric (supports GET, POST, PUT, DELETE), and

although it can be used to compress HTTP interfaces it offers additional functionalities such as built-in discovery, eventing, multicast support and asynchronous message exchanges. From the security point of view several approaches are supported ranging from no-security up to certificate-based one using DTLS. IANA has assigned the port number 5683 and the service name "CoAP".

Within the IMC-AESOP project, CoAP is mainly considered for being used to get access to extremely resource constraint devices, e.g., a temperature sensor, a wireless sensor node, etc. Moreover, the devices may also be mobile and rely on a battery for their operation. These distributed devices would probably be used mostly for monitoring and management, while their integration may enhance the quality of information reaching SCADA/DCS systems.

4.2.4 OPC-UA

One of the challenges in process industries is the interoperability between the systems and devices coming from numerous vendors. This has been addressed by using open standards, enabling devices from different vendors to understand each other. One of the widely accepted standards is OPC (OLE for process control). However, after the years of its use, some limitations of this standard have been evident. This was the reason why OPC Foundation started to work on the new standard – OPC Unified Architecture (OPC-UA) [16].

OPC-UA main improvements over the classic OPC include the following:

- Unified access to existing OPC data models (OPC DA, OPC HDA, OPC A/E, etc.)
- Multi-platform implementations
- Communication and security (OPC has been based on COM/DCOM)
- Data modelling

While the communication, security and interoperability features make OPC-UA great candidate to be used in SOA based applications, it is its data modelling capabilities that enable to build a service oriented process control systems [24]. OPC-UA provides means to access not only the data from the process systems, but also semantic information that is related to the data, like models of the devices that are providing this data. Such models are built by defining Nodes (described by attributes) and Relations between the Nodes.

An information model contains definition of types, from simple to complex, and also instances of such types. The information models are organized and ex-posed by address spaces. In an existing implementation, multiple information models can be defined, for each level on the process there can be a different model of the process entities, however these models can share some information and usually are synchronized. With growing penetration of OPC-UA to the processes and its features that have been designed with Service-Oriented Architecture in mind, it is clear that OPC-UA will become a solid part of service oriented distributed control systems.

4.3 Technology Combinations and Advanced Concepts

Apart from the basic technologies, we take a closer look to efforts for their convergence and provision of more advanced functionalities for future industrial automation systems.

4.3.1 The Embedded Service Framework (ESF)

ESF is a redesigned, rewritten and extended version of the DPWSCore stack, which is available at http://forge.SOA4d.org. The goals of this new version are:

- To bring the power of recent Web-oriented technologies to embedded devices utilizing service-oriented and REST architectures.
- To hide complexity from developers, through code generation and high-level APIs.
- To support a large range of applications, from basic Web applications to complex Web services applications, featuring mechanisms such as network discovery and event publishing.
- To support a wide range of platforms, from mono-threaded (or OS-less), deeply embedded devices (e.g. wireless sensors) to complex multi-threaded applications running on large devices, workstations or enterprise servers.

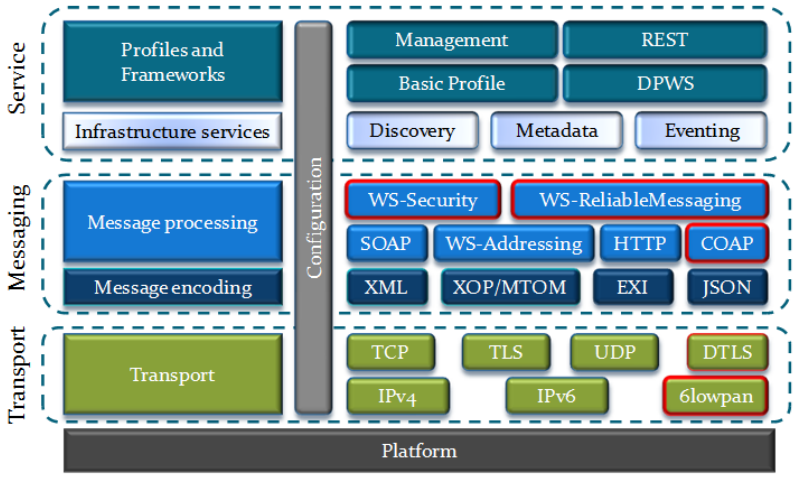

Fig. 4.4 ESF modular architecture

The main features of the ESF include:

- Support of standard IPv4 and IPv6-based transport protocols: TCP, UDP, TLS. Support for additional protocols such as 6LoWPAN and DTLS is also planned.
- Support of standard message encodings: besides XML, which is used in several standard messaging protocols, ESF also supports XOP/MTOM, used to transport binary data in SOAP messages, EXI, a standard binary format for XML well-adapted to low-bandwidth networks, and JSON, a popular format used in particular in browser-based applications.
- Support of several messaging protocols, including HTTP, SOAP (directly over TCP or UDP or combined with HTTP) and SOAP extensions such as WS-Addressing. Planned additions include WS-Security and WS-ReliableMessaging, or CoAP, a draft IETF standard designed for REST applications over 6LoWPAN, but also usable over standard IP networks.
- A set of infrastructure services for network discovery of devices and services, metadata exchange, event publication and subscription or resource management. These services allow the implementation of standard profiles such as Basic Profile (1.1 and 2.0) and Devices Profile for Web services (DPWS), or of resource management frameworks such as WS-Management or ad-hoc solutions based on the REST paradigm.
- A configuration mechanism allowing developers to select the appropriate components from the above list for their applications.

The set of technologies, profiles and frameworks shown in the diagram is not exhaustive: the ESF is extensible and may be used to implement other popular profiles, such as UPnP or ONVIF. Application development on top of the ESF combines access to the runtime library through the ESF API and use of generated code (as shown in). Both client-side and server-side applications may be developed, both sides being often combined in devices capable of peer-to-peer interactions.

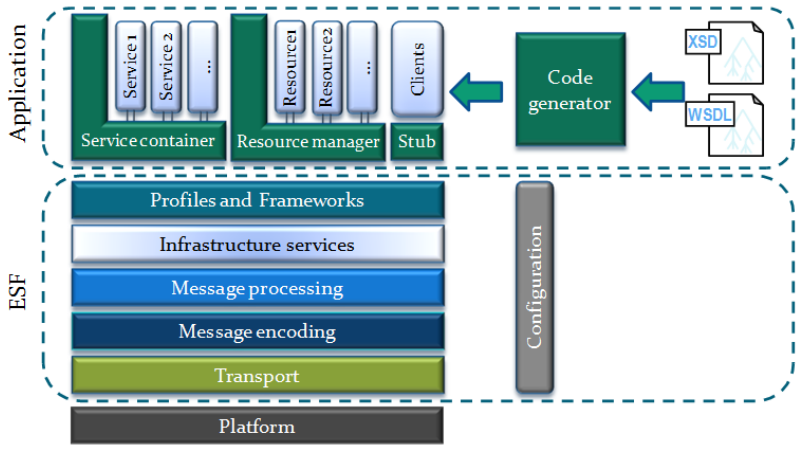

Fig. 4.5 Application development with ESF

On the server side, two paradigms are supported:

- The service-oriented paradigm: based on the abstract definition of a service interface, through a WSDL document, the code generator produces a service skeleton ready to be plugged in the ESF service container. The role of the developer is to provide the implementation of the service operations, and to configure the ESF runtime with the required protocols. ESF makes it easy to publish simultaneously the same services using different protocols, in order to extend the reach of those services to a wide range of clients.
- The resource-oriented paradigm: the ESF provides a resource manager that allows developers to register resource implementations, and that provides remote access to those resources through REST or Web services protocols.

On the client side, code generation is used to produce service stubs, which can be used by the application to invoke remote service operations or resource access. The configurability of the ESF protocols allows clients to access a wide range of devices.

In order to use EXI while guaranteeing the same level of interoperability as XML, several approaches can be considered:

- *Use of a globally shared configuration*: in stable and managed environments, it is possible to deploy the same set of XML schemas in the server and all clients. This single EXI configuration is then used to encode and decode all exchanged messages. This approach has the drawback of being slightly less efficient, as the set of global elements which is used to encode a given message is larger than needed. On the other hand, it simplifies the configuration of the server and the clients.
- *Use of out-of-band information to select the appropriate EXI configuration*: on the server side, it is possible to use external information, such as the HTTP request URL (e.g. when using SOAP-over-HTTP) or the network listener port (e.g. when using SOAP-over-UDP) to select the appropriate EXI configuration. A typical deployment configuration would use different HTTP endpoints for different services and SOAP bindings, and associate to each endpoint the minimal set of XML Schemas needed to parse the incoming EXI messages.
- *Use of EXI options*: EXI provides an in-line header mechanism which allows additional data to be communicated to the EXI processor before it starts decoding, among which a Schema ID. By defining a system-wide naming mechanism for EXI configurations, it is possible to use this solution to dynamically select the appropriate configuration to be used for a given message. This approach has the drawback of slightly increasing the message size, as the Schema ID is systematically embedded at the beginning of each message.

4.3.2 Fusion of DPWS and OPC-UA

As OPC-UA and DPWS have a large set of similarities, it is possible to build a common stack compliant with both standards where the two technologies can benefit from each others. A component implementing the convergence between OPC-UA and DPWS for embedded devices has been prototyped as described in Fig. 4.6.

This component includes in particular:

- The OPC-UA stack developed in ANSI C language by the OPC Foundation and which supports the UA binary profiles defined by the OPC-UA specification.
- The DPWS stack for implementing the Web services profile of OPC-UA.

The architecture of the component makes it possible to change the different libraries of the UA stack to decide which protocol should be supported or not. For this purpose, the server application doesn't need to be changed; only the XML Configuration file must include the good endpoint.

Another goal of this component is to provide a dual interface (i.e. DPWS + OPC-UA). The DPWS and the OPC-UA interfaces are sharing the same data, managed by a Node Manager which contains the implementation of an OPC-UA enabled data model, also called address space.

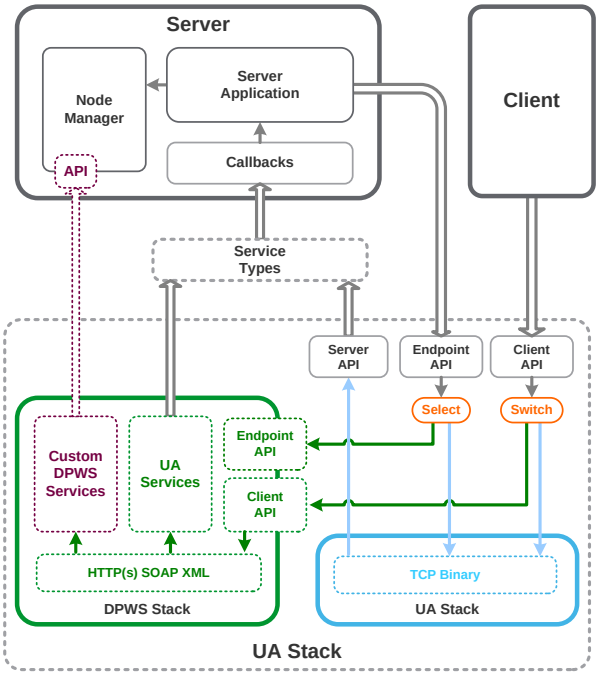

Fig. 4.6 Fusion of OPC-UA and DPWS Architecture

The stack itself mainly consists in the OPC-UA and DPWS parts and a unified API between the server or client application and the stack. The key parts are:

- *OPC-UA part*: Contains the marshalling and de-marshalling code for the UA binary protocol (over TCP). It also contains the standard definitions, data structures and data types for OPC-UA (some parts are partly used in the DPWS part to get a binding between DPWS and OPC-UA).
- *DPWS part*: Contains the marshalling and de-marshalling code for the UA SOAP XML protocol (over HTTP and HTTPS) and for the implementation of other service operations based on DPWS (Custom DPWS Services).
- *Server API*: Represents the interface to the server application to react to incoming messages from a client requested over the different service operations.
- *Endpoint API*: Represents the interface for the server application to manage endpoints (Create, Open, Close, Delete, ...).
- *Endpoint API for DPWS part*: Represents the interface for the DPWS stack to manage endpoints (internal API). The design is related to Endpoint API of the final stack which can be called from outside.
- *Client API*: Represents the interface for the client application to use service operations for the communication with a server.
- *Client API for DPWS part*: Represents the interface for the DPWS stack for using the supported client operations (internal API).
- *Service Types*: Responsible to call the correct callback function in the server application concerning the called service operation from a client. More information about the service types is given in the following chapter.

The following features have been implemented and tested, showing that the DPWS stack can be used for implementing a HTTP/HTTPS profile for an OPC-UA stack and that the resulting component can expose both an OPC-UA and a DPWS interface:

- Communication over HTTP SOAP XML profile is working
- Communication over OPC TCP Binary profile is working
- Communication over HTTPS is working
- Server can be used to deploy a predefined XML data model description for a device
- Custom Web services can be discovered and called in conformance with the DPWS specification

The DPWS / OPC-UA prototype has demonstrated promising benefits for systems with a large number of devices in particular when the data exposed by the devices are heterogeneous. In the following we consider a system including a client application and a set of devices or sub-systems where all communicating entities are implementing a converged stack with DPWS and OPC-UA: DPWS brings the capability for the client application to discover dynamically a large amount of devices. We have tested that at least 1000 devices can be discovered at the same time.

The discovered devices have then to expose their data to the client application. Even if the semantics of the data exposed by all the devices are heterogeneous, the

data can be individually mapped on the generic meta-model of OPC-UA. This can be done through proprietary mappings or preferably by mappings already specified and validated by the OPC Foundation (OPC-UA companion standards). For the OPC-UA enabled client application, the result is that it can understand the data exposed by all the devices. This client application may be either a completely generic OPC-UA application, in which case it will understand the data with a limited semantic level, or the client application may be more aware of the domain semantic (either proprietary or defined in OPC-UA companion standards).

4.3.3 Distributed Service Bus

Web service based technologies investigated so far at device level (DPWS, OPC-UA, etc.) rely mainly on point to point communication models, which do not favour the system scalability. The "Service Bus" approach aims at decoupling service consumers from service producers in the industrial process control system. Large scale distributed systems can benefit from a Service Bus type middleware architecture as the bus acts as a broker between the numerous service consumers/providers, avoiding a potentially huge number of point to point connections.

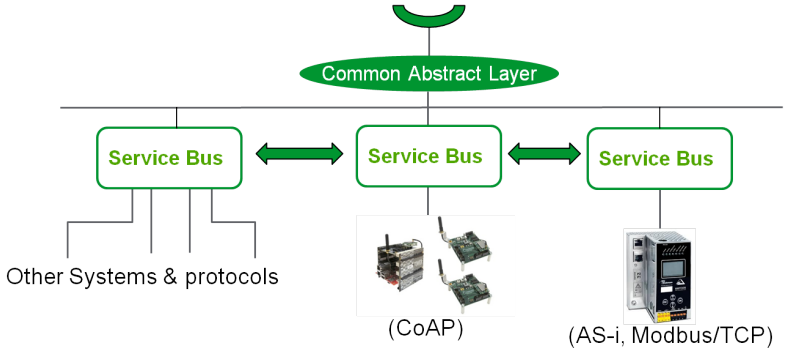

Fig. 4.7 Using the Service Bus as Common Abstraction Layer

The Service Bus middleware (depicted in Fig. 4.7) is based on a distributed architecture in order to share information between all middleware instances. In other terms, devices and systems handled by an instance of the Service Bus are exposed through a normalized data model and this information is shared with the other instances.

Legacy systems can also benefit from service bus architecture as the bus acts as a gateway between legacy systems and IMC-AESOP SOA systems. This service bus is therefore the natural place for adding a semantic layer on top of legacy services. Thus, the bus provides an abstraction of technical devices and services into business oriented/domain specific services descriptions.

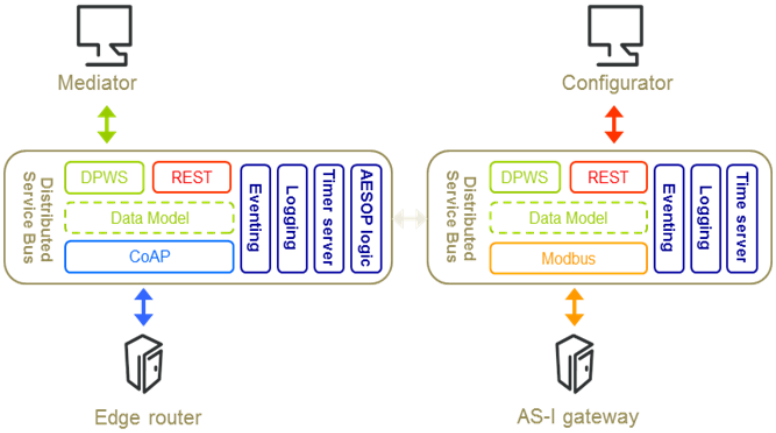

Fig. 4.8 Distributed Service Bus Architecture

Fig. 4.8 gives a functional view of the distributed service bus and illustrates how it hosts some of the services identified in the IMC-AESOP architecture study, for instance:

- Gateway functionality through a variety of connectors;
- Registry as a central repository for IMC-AESOP services;
- Code/configuration/model repository (not implemented yet);
- Event broker for true loose-coupling between event producers and consumers;
- Security services;
- DNS service;
- Historian/logger;
- Time service for time synchronisation between IMC-AESOP services;
- Native interface (Web services) to higher level information systems (MES/ERP...).

The modularity of the service bus allows adding protocol connectors and application modules, in order for instance to manage various devices and services. Such management operations are applied through a common abstract layer. The distributed architecture of the Service Bus allows a management operation to be routed to the adequate Service Bus instance handling the targeted device or service. Therefore, the distributed architecture of the Service Bus and the common interface through the abstract layer both enable the management of large scale systems.

The Service Bus implementation is currently available in C and Java languages. The C brick can be embedded in devices with constrained resources, it requires around 200 KB of Flash memory and 50 KB of RAM with all connectors/modules included. The Java brick requires obviously much more resources and will run on more powerful devices able to run a Virtual Machine. This is the gateways and controllers that can be found in typical process control systems. The Java implementation relies on the OSGi Framework "Felix" (felix.apache.org). OSGi is an

SOA-based modular framework which implements a dynamic component model. It also provides dynamic lifecycle management for its modules which can be started, stopped, updated without an application reboot. This capability is particularly interesting for high end Service Bus instances, where new modules and connectors can be deployed at runtime.

The Service Bus provides several set of management operations which can be applied to devices and services. Device Management capabilities include adding/removing/discovering devices, getting/setting configuration and status. Typical service management includes start/stop/reset of services, getting/setting configurations and status. The device and service management operations are accessible through both SOAP and REST interfaces.

The distribution among Service Buses is also handled through an internal DPWS/SOAP interface. This DPWS interface handles the mutual discovery between Service Bus instances thanks to WS-Discovery.

Time synchronisation relies on the IEEE 1588 PTP (Precision Time Protocol) protocol running on all Service Bus instances. Time synchronisation is a strong requirement for events timestamping and correlation. Events logging is implemented based on the standard syslog protocol, which allows to aggregate events from all Service Bus instances on a central repository. This is particularly useful for correlating system events, for root cause analysis for instance. This capability has been used in particular in use case 1, providing useful insights on system behaviour.

Cyber-security was not at the heart of the IMC-AESOP project so only a minimal support was provided through the Service Bus component. This includes user authentication through HTTP basic authentication and a simplified Role Based Access Control (RBAC) applied to each service call. Practically, only admin users were able to invoke services from the Service Bus.

The Service Bus can handle large-scale distribution by relying on its connectors distribution. Each connector exposes Devices and Services in a common abstract way. Moreover, information from the abstract layer is actually exchanged between all instances of the Service Bus. Such distribution allows any application to interact with the real devices/services transparently through a common interface which is provided by any Service Bus instance.

4.3.4 Complex Event Processing (CEP)

Throughout the last years, Complex Event Processing (CEP) [15] has gained considerable importance as a means to extract information from distributed event-based (or message-based) systems. It became popular in the domain of business process management but is now being applied in the industrial monitoring and control domains. It is a technology to derive higher-level information out of low-level events. CEP relies on a set of tools and techniques for analysing and handling events with very low latency. The feature set for CEP spans from event extraction, sampling, fil-

tering correlation and aggregation to event enrichment, content based routing, event compositions (and not only limited to these).

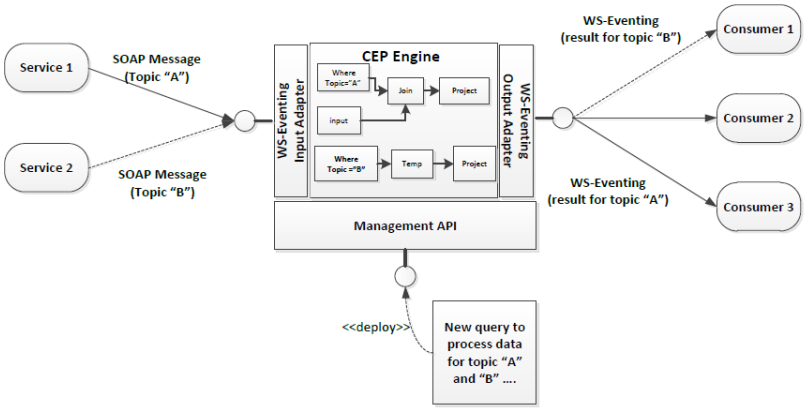

Fig. 4.9 Complex Event processing mechanism in a SOA-Infrastructure

Originally CEP systems were created at enterprise systems; therefore, most of available systems are providing tools to define queries and to manage and administrate the system. Some of them are providing concepts for scalability and resilience. In contrast, nowadays, we can observe a trend to move CEP closer to the place where the data is born to enable early filtering, aggregation and resampling capabilities. In that way it will become possible to write or define a query and distribute it seamless cross a distributed setup in a way to reduce network traffic and save bandwidth.

Normally, complex events are created by abstracting from low-level events. The processing of events is expressed within a specific language in terms of rules. Unfortunately, the set of features and the way to express the rules differ from platform to platform. CEP engines are able to process events up to 100,000 events/sec. This clearly depends on the complexity of the rules. Normally the limitation is set by the connection to the external environment, such as extraction of events from input sources or the limitation by the bandwidth of the network.

So far, there is no unified way to express rules (or queries) over streams of events. Thus, it makes sense to wrap a CEP engine (Fig. 4.9) within a service with well-defined endpoints. The endpoints are technology agnostic and define the operations and data to be processed while the CEP service itself is responsible for transforming the data/messages to its internal event format. On the output side consumers can subscribe via WS-Eventing so that notifications can be sent via SOAP messages as well (see Fig. 4.8). This approach enables the integration [7] with specifications like Device Profile for Web services (DWPS) and OPC Unified Architecture (OPC-UA), which are the most suitable solutions for implementing a SOA since both specifications include eventing mechanisms.

Two kinds of CEP are expected to be provided in future industrial systems:

- *CEP as a service*: When we are saying service, it means that this can either be realized as a service running locally on a server or the same concept still holds for a service running in the cloud and on top of cloud technologies.
- *Embedded CEP*: This is a concept of a lightweight CEP using concurrent reactive objects (CRO) model guaranteeing execution of CEP queries in an efficient and predictable manner on resource constrained platforms and offering a low-overhead real-time scheduling.

By enabling event processing mechanisms IMC-AESOP is also considering the convergence of scan-based and event-based mechanisms. This is achieved by supporting pull- or push- models [22]. The services can either send events (active) to the CEP service or there is a mediator which pulls data form services (passive) and sends this data. From the CEP service this looks like an active data service provider. On the output side results are pushed to registered consumers.

4.3.5 Semantic-driven interaction

Enabling interoperability of the service specifications and data models is a key technological challenge that SOA systems are aimed to resolve. The full interoperability requires that the syntax and semantic service descriptions are well defined, unambiguous and enable dynamic discovery and composition. Thus far, most if not all SOA installations are enabling pure syntax interoperability with little or no support for standard based semantic descriptions. The use of structured data formats only partially resolves the problem by supplementing the exchanged data with meta-information in the form of tags and attributes in the case of XML/EXI for example. The tag names are ambiguous and usually insufficient to describe the service functionality in full.

Applying application level data model standards is often a remedy as the syntax to semantics mapping is predefined. Example of such standard is Smart Energy Profile 2 that clearly states the physical meaning of the tag names and structures defined for the service messages in the domain of energy management. One problem when complying with such standards is that they are almost always domain specific which requires mapping of the semantic descriptions from one standard to all others in use.

Another approach is to define generic semantic data model that is applicable to wide range of use cases. The initial investigation highlighted the Sensor Model Language (SensorML) [20] as a promising specification for generic semantic description of sensory data. However, the complexity and size of SensorML specification limit its use to more capable devices. Small scale experiments with a number of sample SensorML messages showed that even EXI representation will not be sufficiently small to fit a battery powered wireless sensor nodes that have low-power, low-bandwidth radios.

Another possible specification for sensor data is the Sensor Markup Language (SenML) [9]. It has a very simple design that is consistent with RESTful architec-

ture and is targeted at resource-constrained devices. An initial evaluation of SenML revealed that it meets the requirements for hardware utilisation but there are areas that are too much simplified and insufficient to describe the data in the details required by the applications. An example of such limitation is the precision of the time stamping of the sensor data – SenML allows for up to seconds resolution that is not enough for many industrial use cases. To overcome this limitation, we had to use a custom generic data representation that is reusing many of the design choices in SenML.

4.4 Discussion

DPWS, coming from the IT world, is the most applicable set of Web services protocols, to be used at the device level. Combined with EXI, it provides real capabilities in the range of the millisecond, following the technology assessment made by the project. OPC-UA, coming from the industrial world, is also a set of Web services protocols, compatible with DPWS, and providing a data model enlarging the semantic capabilities of the solution. CoAP can be used for wireless sensor networks. It can also be combined with EXI. This is still work in progress with major impact in the future once the technology matures.

The Service Bus and the CEP solution are technologies providing the large scale and migration capabilities, combining and processing information coming through DPWS, OPC-UA or legacy protocols, in order to manage large-scale event-based systems. A suitable combination of the six technologies described above is able to provide solutions meeting the four critical questions and challenges expressed in the introduction.

After some initial assessment and taking into considerations the operational context of IMC-AESOP, we have come up with a synthesis of the most promising technologies (depicted in Table 4.1), which are being used to implement the IMC-AESOP prototypes:

Table 4.1 Technologies and Challenges

Technologies	**Real-time**	**Management of large scale**	**Event-driven**	**Semantics**
DPWS		X	X	
OPC-UA		X		X
CoAP		X	X	
EXI	X		X	
Service Bus		X		
CEP			X	

A closer look among some of the Web services based technologies and their performance [11] reveals several aspects. Although DPWS is already available and supported by several devices in multiple domains, we can clearly see that in its

standard form it has a significant impact on computational and communication resources. Hence devices that may consider this stack, should be usually devices that are on the upper scale with respect to their resource availability. REST and CoAP are designed for much more lean environments and as we see these are a much better fit for resource constrained devices e.g. in comparison to DPWS [18]. Additional combinations of DPWS with compression techniques however could remove this barrier [17].

REST and CoAP approaches are more lightweight (from CPU and memory utilisation) and more user-friendly implementation-wise, and therefore could empower even simple sensors to take part in the "Cloud of Things" [11]. On the cloud side, since we do not have significant resource problems, any of the stacks can be used, but maybe for scalability reasons the lightweight REST might also be preferred, unless some specific functionality is needed, e.g. WS-Discovery from the DPWS in order to dynamically discover embedded devices and their services. Further customisations may enable hybrid approaches such as SOAP over CoAP [19]. Additionally ongoing work e.g. in EXI [3] may also enable better performance when combined with the XML based approaches.

OPC-UA is not a real-time protocol, but is designed rather to gather information about the transferred data with the occurrence time stamp and distribute that information on demand [4]. OPC-UA services are designed for bulk operations to avoid roundtrips, something that increases the complexity of the services but greatly improves the performance [16]. Nevertheless, the balance between functionalities and performance needs to be per scenario investigated, especially due to the multiple other aspects the OPC-UA brings with it as already analysed.

Although the initial tests [11] are not conclusive and offer only a notion of performance, there are several other issues that need to be investigated and which may be of critical importance, depending on the application domain targeted. Security is an issue, and the impact has not been investigated here as we considered only *HTTP* calls. The impact also of *HTTP pipelining* as well as new future Internet *HTTP*-modified networking protocols like *SPDY* [1] and *HTTP Speed+Mobility* [23] that offer reduced latency through compression, multiplexing, and prioritisation need to be assessed. Additionally, other issues such as excess buffering of packets may cause high latency and jitter [5], and this may have significant impact on network performance, which might be a show-stopper e.g. for time-critical applications.

4.5 Conclusions

We have attempted to tackle four major critical questions that arise as key when technologies have to be selected and used to implement a Service-Oriented Architecture (SOA) based distributed large scale process monitoring and control system. After compiling and assessing a set of technologies, a subset of them has been selected and used by the IMC-AESOP consortium. It is important to call the attention to the fact that the selected technologies, are either already available from open-source

sites or they are still under development by some of the IMC-AESOP technology-provider partners.

Following the assessment of the prototype implementations, which refined the technology evaluation and investigate other challenges in implementing SOA-based cross-domain infrastructures, e.g. cloud of services generated from the virtualisation of different systems like manufacturing, smart grid, transportation, etc. the experimentation results have shown that technological choices made are quite promising for next generation SCADA/DCS systems:

- For real-time SOA, EXI, the binary XML format, makes a lot of sense for wireless interconnections, in particular for CoAP based data exchanges even though CoAP does not support true real-time capabilities in the current design. The benefit of EXI compression is also less obvious for wired SOAP based Web services.
- Proposing a dual OPC-UA/DPWS stack can facilitate the management of large scale distributed systems by building a bridge between industrial automation and IT worlds.
- A SOA middleware like the proposed Distributed Service Bus can ease the integration and interoperability of heterogeneous technologies in the plant.

Acknowledgment

The authors would like to thank the European Commission for their support, and the partners of the EU FP7 project IMC-AESOP (www.imc-aesop.eu) for the fruitful discussions.

References

[1] Belshe M, Peon R (2012) SPDY Protocol. IETF Internet-Draft, URL http://tools.ietf.org/html/draft-mbelshe-httpbis-spdy-00

[2] Bormann C, Castellani AP, Shelby Z (2012) Coap: An application protocol for billions of tiny internet nodes. IEEE Internet Computing 16(2):62–67, DOI http://doi.ieeecomputersociety.org/10.1109/MIC.2012.29

[3] Castellani A, Gheda M, Bui N, Rossi M, Zorzi M (2011) Web Services for the Internet of Things through CoAP and EXI. In: Communications Workshops (ICC), 2011 IEEE International Conference on

[4] Fojcik M, Folkert K (2012) Introduction to opc ua performance. In: Kwiecień A, Gaj P, Stera P (eds) Computer Networks, Communications in Computer and Information Science, vol 291, Springer Berlin Heidelberg, pp 261–270, DOI 10.1007/978-3-642-31217-5_28, URL http://dx.doi.org/10.1007/978-3-642-31217-5_28

[5] Gettys J, Nichols K (2012) Bufferbloat: dark buffers in the internet. Commun ACM 55(1):57–65, DOI 10.1145/2063176.2063196, URL http://doi.acm.org/10.1145/2063176.2063196
[6] Hilbrich R (2010) An evaluation of the performance of dpws on embedded devices in a body area network. In: Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE 24th International Conference on, pp 520–525, DOI 10.1109/WAINA.2010.93
[7] Izaguirre M, Lobov A, Lastra J (2011) Opc-ua and dpws interoperability for factory floor monitoring using complex event processing. In: Industrial Informatics (INDIN), 2011 9th IEEE International Conference on, pp 205–211, DOI 10.1109/INDIN.2011.6034874
[8] Jammes F, Bony B, Nappey P, Colombo AW, Delsing J, Eliasson J, Kyusakov R, Karnouskos S, Stluka P, Tilly M (2012) Technologies for SOA-based distributed large scale process monitoring and control systems. In: 38th Annual Conference of the IEEE Industrial Electronics Society (IECON 2012), Montréal, Canada.
[9] Jennings C, Shelby Z, Arkko J (2013) Media types for sensor markup language (SENML). Tech. rep., IETF Secretariat, URL http://tools.ietf.org/html/draft-jennings-senml-10
[10] Karnouskos S, Colombo AW (2011) Architecting the next generation of service-based SCADA/DCS system of systems. In: 37th Annual Conference of the IEEE Industrial Electronics Society (IECON 2011), Melbourne, Australia.
[11] Karnouskos S, Somlev V (2013) Performance assessment of integration in the cloud of things via web services. In: IEEE International Conference on Industrial Technology (ICIT 2013), Cape Town, South Africa
[12] Karnouskos S, Colombo AW, Jammes F, Delsing J, Bangemann T (2010) Towards an architecture for service-oriented process monitoring and control. In: 36th Annual Conference of the IEEE Industrial Electronics Society (IECON-2010), Phoenix, AZ.
[13] Karnouskos S, Colombo AW, Bangemann T, Manninen K, Camp R, Tilly M, Stluka P, Jammes F, Delsing J, Eliasson J (2012) A SOA-based architecture for empowering future collaborative cloud-based industrial automation. In: 38th Annual Conference of the IEEE Industrial Electronics Society (IECON 2012), Montréal, Canada.
[14] Kyusakov R, Eliasson J, Delsing J (2011) Efficient structured data processing for web service enabled shop floor devices. In: Industrial Electronics (ISIE), 2011 IEEE International Symposium on, pp 1716–1721, DOI 10.1109/ISIE.2011.5984320
[15] Luckham DC (2001) The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA
[16] Mahnke W, Leitner SH, Damm M (2009) OPC Unified Architecture. Springer
[17] Moritz G, Timmermann D, Stoll R, Golatowski F (2010) Encoding and compression for the devices profile for web services. In: 24th IEEE International

Conference on Advanced Information Networking and Applications Workshops, WAINA 2010, Perth, Australia

[18] Moritz G, Zeeb E, Prüter S, Golatowski F, Timmermann D, Stoll R (2010) Devices Profile for Web Services and the REST. In: 8th International Conference on Industrial Informatics (INDIN), Osaka, Japan.

[19] Moritz G, Golatowski F, Timmermann D (2011) A lightweight SOAP over CoAP transport binding for resource constraint networks. In: Mobile Adhoc and Sensor Systems (MASS), 2011 IEEE 8th International Conference on, pp 861 –866, DOI 10.1109/MASS.2011.101

[20] OCG (2007) Sensor Model Language (SensorML) Implementation Specification. URL http://www.opengeospatial.org/standards/sensorml

[21] Shelby Z (2010) Embedded web services. Wireless Commun 17(6):52–57, DOI 10.1109/MWC.2010.5675778, URL http://dx.doi.org/10.1109/MWC.2010.5675778

[22] Tilly M, Reiff-Marganiec S (2011) Matching customer requests to service offerings in real-time. In: Proceedings of the 2011 ACM Symposium on Applied Computing, ACM, New York, NY, USA, SAC '11, pp 456–461, DOI 10.1145/1982185.1982285, URL http://doi.acm.org/10.1145/1982185.1982285

[23] Trace R, Foresti A, Singhal S, Mazahir O, Nielsen HF, Raymor B, Rao R, Montenegro G (2012) HTTP Speed+Mobility. IETF Internet-Draft, URL http://tools.ietf.org/html/draft-montenegro-httpbis-speed-mobility-02

[24] Trnka P, Kodet P, Havlena V (2012) Opc-ua information model for large-scale process control applications. In: IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society, pp 5793–5798, DOI 10.1109/IECON.2012.6389038