

Accessing Fieldbus Systems via Web Services

Marcus Venzke¹ and Stefan Pitzek²

¹Telematics Department,
Technische Universität Hamburg-Harburg, Hamburg, Deutschland
venzke@tu-harburg.de

²Real Time Systems Group,
Vienna University of Technology, Vienna, Austria
pitzek@vmars.tuwien.ac.at

***Abstract** — This paper discusses accessing fieldbus systems via web services over the Internet or intranets, having simplicity and interoperability as advantages. For this purpose an architecture is proposed, in which web service implementations connect to gateway nodes of fieldbus systems. Three use cases show the potential of this approach for configuration, management, and non-real time control. Intelligence to protect fieldbus systems against compromising accesses from the Internet or an intranet is provided as the so-called automatic validation. It rejects all accesses not conforming to the specification of the web service's interface.*

1 Introduction

In many industry fields there is a notable tendency for integrating existing business infrastructures into networks, such as the Internet. This general approach also holds true for fieldbus and smart transducer¹ networks, which also get a tighter integration into the existing higher-level network structure. Due to different communication properties of fieldbus and high-level networks, usually only a subset of available functions and operations can be provided to the higher-level networks. For example, fieldbus networks often exhibit deterministic *real-time* behavior, whereas the Internet takes a best-effort approach when transmitting data. Operations that often can be provided over the high-level network include monitoring, configuration and maintenance.

At the boundary between a fieldbus and a high-level network we usually find a gateway node, which acts as the mediator of communication between the connected networks. While the general approach is quite common, often no further specifications are provided how the actual access from the higher-level network to the fieldbus system is performed. Accordingly, in many cases the fieldbus access interface will be specified in an ad hoc

¹ For Smart Transducers see [1].

manner, which contradicts the interoperability idea, which often was one of the major reasons for integrating networks into interconnected networks in the first place.

There have been some efforts for standardising the access to fieldbus systems. For example the OMG recently accepted a *smart transducer interface* as a world-wide IOP standard [4], which specifies a general CORBA interface for interacting with fieldbus systems. The Field Device Tool (FDT) Specification (as used for example in the Profibus [3]) also provides an interface between the fieldbus (the control application) and a higher-level network by specifying interfaces, which can be accessed via the distributed component object model (DCOM) [1].

Another common way for interacting with fieldbus systems are gateway nodes, which act as web servers, thus supporting access to fieldbus systems via the Internet using a web browser. In this approach, the application logic is completely contained in the web application and the fieldbus system. Clients, containing additional application logic, accessing the fieldbus system via the web server, are not possible.

This is enabled with web services, a promising technology for providing software interfaces over the Web. Web services allow a simple and interoperable form of communication between different platforms. Unlike related technologies such as CORBA, web services are intended to provide software interfaces (to services) for global access. These properties make them a befitting basis for the integration of fieldbus systems with external tools like intelligent configuration, production planning and machine control systems.

In the paper we analyse this type of integration and highlight potential benefits. We propose an architecture for accessing fieldbus systems via web services. By examining exemplary use cases we show the potential of using such type of high-level Internet-based communication, addressing issues of configuration, management and non-real time control. We also present an extension of the architecture, adding intelligence to protect fieldbus systems and its web service implementation against messages that do not conform to their specification. In order to highlight the possibilities and issues, we discuss some of the ideas in the context of the TTP/A fieldbus. TTP/A has been chosen because of its simplicity and well-defined interfaces.

The paper is structured as follows. Section 2 provides an overview on TTP/A fieldbus, section 3 introduces to web services. The architecture for accessing fieldbus systems via web services is presented in section 4, for which use cases are shown in section 5. Section 6 introduces to the specification technique SXQT for web service interfaces. Protection against non-conforming message flows can be achieved with the automatic validation described in section 7. Section 8 concludes the paper.

2 Smart Transducer Interface

The time-triggered protocol for SAE Class A applications (TTP/A) provides a time-triggered fieldbus primarily intended for low-cost applications in the automation and automotive area. TTP/A follows the interface definitions as specified in the OMG smart transducer interface standard. All accesses to the fieldbus are performed via three conceptual interfaces [5]:

Real-time Service (RS) Interface: Responsible for providing a timely real-time service of the smart transducer during operation.

Diagnostic and Maintenance (DM) Interface: Enables the access of internals of the smart transducer. This is required to set parameters, or gain information on internals of the node, which could be useful, e.g., for diagnosis purposes. Generally the DM interface is not considered time-critical.

Configuration and Planning (CP) Interface: Enables access to configuration properties of a node. During system start-up and node integration the interface is used to provide connecting information between multiple nodes for enabling their interaction. Generally the CP interface is also not considered time-critical.

All accesses are mapped to the interface file system (IFS), which acts as a temporal firewall between the node application and the communication network, i.e. it completely decouples the node application from the communication activities. The IFS is a file system abstraction supporting read, write and execute operations on parts of the file system. All information that is directly relevant for the services of a node is placed into the IFS. This also includes the so-called round descriptor lists (RODL), which define the strictly time-triggered communication schedule of a TTP/A node.

Since the smart transducer interface and the IFS define a simple access and consistent interface to fieldbus nodes, it provides an ideal point of access for creating the connection between the specification of the web services and the actual content of the IFS. These descriptions are themselves represented with XML, so that a further integration with XML-based web services seems quite natural.

3 Web Services

Web services allow the creation of interoperable software interfaces using web technology. Combining the standard encoding XML for messages and the well-understood transport protocol HTTP into the protocol SOAP [6], they stand for simplicity and interoperability between different platforms.

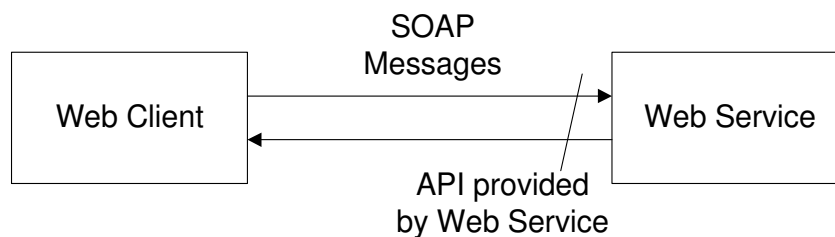


Figure 1: Communication between web client and web service

Web services are web applications that are accessible by software clients (web clients). Traditional web applications have proven their abilities to allow application access in an interoperable manner on the worldwide Internet and enterprise-wide intranets. However the access is restricted to humans using web browsers, mainly due to the use of HTML combining data with formatting information. Web services resolve this issue by replacing HTML with XML for request and response messages transmitted with HTTP. The protocol SOAP adds conventions for the message exchange and the structuring of messages (SOAP messages). This mode of interaction is illustrated in Figure 1.

The Web Services Description Language (WSDL) [7] is used to describe the interfaces provided by web services. It describes interfaces in terms of exchanged messages. Interfaces are defined as sets of operations, which mainly consist of the two message types for its requests and responses. XML fragments in these messages are declared in XML-Schema [8], the common type system for XML.

Using web services to provide interfaces over the web leads to interoperable solutions and does not require complex infrastructures. Implementations can easily be created using widely available infrastructures for web applications plus support for XML. More specialized infrastructures are also available. Their interoperability is permitted describing interfaces with WSDL's clear message exchange semantics. The consensus for web services in industry and research leads to a wide range of tools and applications on different platforms.

4 Web Services for Fieldbus Access

To gain the advantages of simple and interoperable access to fieldbus systems, we propose to provide their interfaces using web services. The required architecture is depicted in Figure 2. The web service implementation provides the interface over the web to be used by external tools. Running outside of the fieldbus system, it connects to it via the proprietary interfaces of a gateway node.

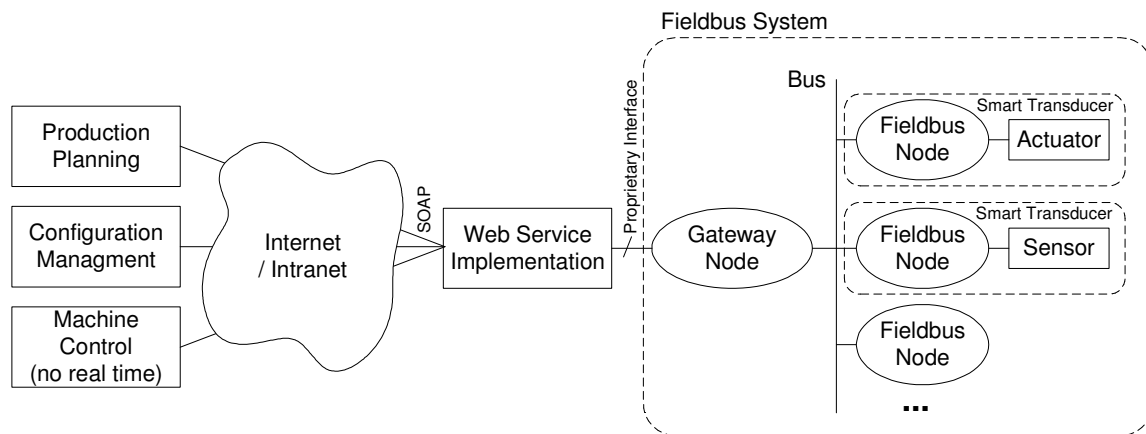


Figure 2: Architecture to access fieldbus system via web service

The provided interfaces can be general purpose or application-specific. A general purpose interface is usable for a variety of different fieldbus systems and applications. Its semantics can be similar to CORBA's smart transducer interface [4], where accesses are mapped to the interface file system (IFS), which allows reading, writing or executing records in its files. Other interfaces might be specifically designed with an application-specific semantics that are easier to use by developers. In this case the web service implementation and/or gateway node contain application-specific code. This approach is similar to providing a web application for use with a web browser, but allows additional application logic in the web client.

Web services usually cannot provide real-time services, because the underlying Internet does not have the required properties. This is no restriction, as long as non time-critical

services are used (i.e. the CP and DM interfaces). For services that normally depend on the Real-time Service (RS) Interface (see chapter 2), such as controlling machines, high-level application-specific interfaces need to be developed that are not time critical. Chapter 5 gives an example how such an interface can be developed.

Calls to web services should be coarse grained. Even though the Internet provides an increasing bandwidth, the latency is still high, leading to a high minimum execution time for operation calls. In case of many operations to be called sequentially this seriously decreases the performance. To improve the performance interfaces should be modelled in a way that minimizes the number of operation calls. For example consider CORBA's smart transducer interface. If this interface would directly be mapped to web services, each access to a 32-bit record would require an individual web service call. Instead an interface of a web service should allow the access a lot of records with one call. The web service implementation then has to split up the call into several calls to the IFS.

5 Web Services Sample Use Cases

Having access to fieldbus systems via web services enables a lot of applications. In the following we present three use cases in different application areas, the third in more detail.

A production planning system (PPS) can control a factory's production line via web services. A PPS is used to plan when to process which purchase order. If every machine in the production line is implemented as a fieldbus system accessible via web service the PPS can directly configure them. The PPS can also detect, if a machine is faulty. It can then automatically reconfigure the production line to replace the faulty machine or to process another purchase order that does not require it.

In the second use case a car manufacturer maintains or configures a car via a web service. The car is a fieldbus system, in which smart transducers contain required sensors and actuators. In a garage the car is connected to a gateway, giving the manufacturer access via a web service over the Internet. The manufacturer's experts can then locate faults and reconfigure the fieldbus system if required. Similar to the previous use case this is an example how web services enable the connection of fieldbus systems with complex configuration systems, which contain the intelligence required to adapt the fieldbus system to changing situations.

The third use case presents a web service for controlling an industrial robot. In general controlling a robot requires real-time interfaces, which cannot be provided by web services. However for some applications it is possible to develop application-specific interfaces that are not time critical. Consider a robot with n articulations, which can be set to defined angles, as illustrated in Figure 3. The vector of required angles $(\omega_1, \omega_2, \dots, \omega_n)$ is the parameter for an operation `SetPositions`, which sets the articulations to these angles. The operation is synchronous, thus the response is not sent before the positioning has finished. Afterwards the operation can safely be called again in a non time critical manner.

Additional operations are added to the interface to allow only one controlling web client at the same time. Before a web client starts to control the robot it has to call the operation `Login`, which fails, if another web client is already logged in. After controlling the robot is completed the operation `Logout` is called.

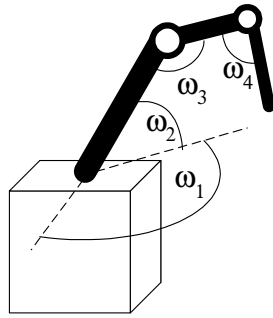


Figure 3: Robot with 4 articulations

A web client using the interface has to obey several requirements. `Login` and `Logout` need to be called in alternating order. `SetPositions` may only be called after `Login` and before `Logout` and must not be called several times concurrently. The combination of angles $\omega_1, \omega_2, \dots, \omega_n$, given to `SetPositions` as parameter, must not harm the robot e.g. by crashing against a wall. Unfortunately these requirements cannot be expressed in WSDL.

6 SXQT

The rigorous specification of requirements as those just described is fundamental for an interoperable, reliable communication over the interface of a web service. Therefore we have developed the expressive and generic specification technique SXQT (Specifications using XQuery expressions on Traces), based on a specification technique of C.A.R Hoare [9], with major enhancements to specify interfaces of web services.

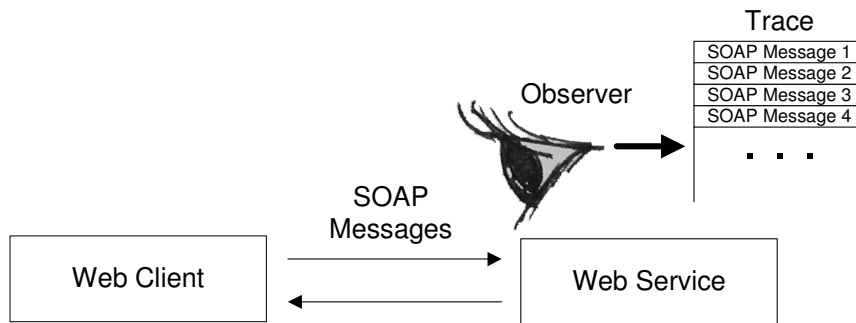


Figure 4: Observer recording a trace

SXQT constrains allowed message flows with first order logic. Observations of messages are considered as atomic events. An observer records them in the order of their observation until some point in time into the so-called trace, as illustrated in Figure 4. A trace thus formalises an observed message flow. Predicates on traces (SXQT expressions) formulated in the standardized language XQuery [10] are used to constrain the allowed message flows. Every requirement on an interface is formulated individually as an SXQT expression, which is added to the description in WSDL.

By using SXQT the message flows can be specified that do not compromise the fieldbus system. It allows the specification of the ordering of operation calls, as described in the robot use case for the operations `Login`, `Logout` and `SetPositions`. It enables the specification of concurrency constraints, such as the absence of concurrent calls of the operation `SetPositions`. It also enables to specify the allowed combinations of values in parameters, e.g. the allowed combination of angles ω_1 , ω_2 , ..., ω_n provided as parameter to the operation `SetPositions`. All this is done with the same generic constructs.

7 Automatic Validation

To check that message flows conform to the specification we have developed a technology called “automatic validation”. As illustrated in Figure 5 a validator implements the conceptual observer from SXQT. It observes all exchanged messages and records them into the trace. In addition it checks if the message flow conforms to the specification. Non-conforming messages are rejected and responded to with an error message. This protects the fieldbus system effectively against compromising message flows. Additionally the automatic validation detects non-conforming messages sent by the web service, immediately indicating precarious conditions in the fieldbus system or the web service implementation.

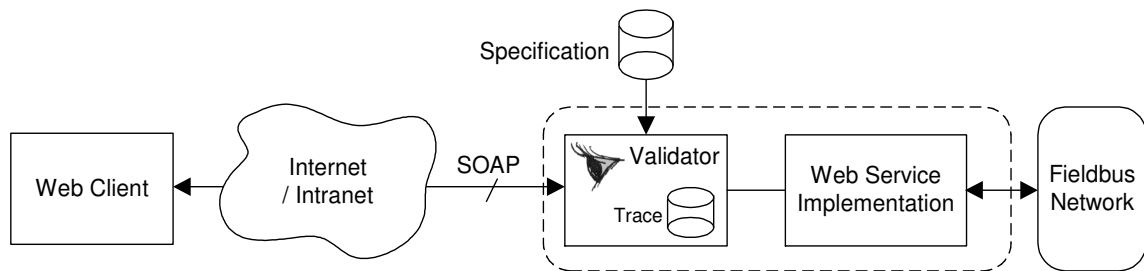


Figure 5: Architecture with validator

The validator reads the specification from a document, containing the description in WSDL and the SXQT expressions. All SXQT expressions are evaluated for the observed trace. In addition for each message it is checked whether it conforms to the WSDL-description and the SOAP standard. A message is rejected, if any of the checks fails.

The validator can run inside the process of the web service implementation or in a separate process. Running in the process of the web service implementation leads to better performance, since messages need to be received and parsed only once and can efficiently be forwarded inside the process. If a validator runs in a separate process and forwards messages to the web service implementation via the network, it can be used universally with any web service.

The validator contains the intelligence required to prevent that non-conforming messages compromise the web service and thus the fieldbus system. The validator rejects non-conforming messages sent by a web client before they are delivered to the web service. Thus the web service will never observe non-conforming messages. In the robot use case the web service can never observe operation calls that are not in the required order or that are breaking the concurrency constraint. Also calls to the operation `SetPositions` with an illegal combination of angles ω_1 , ω_2 , ..., ω_n provided as parameter are concealed.

8 Conclusion

In the paper we have proposed an architecture, which allows accessing fieldbus systems via web services over the Internet or intranets. Software interfaces based on SOAP are provided by web service implementations, which connect to the fieldbus system via the proprietary interfaces of its gateway node. Three use cases in different fields have shown the applicability of this approach.

The third use case on controlling a robot was analysed in more detail. The requirement for real-time interfaces was avoided by providing an application-specific abstraction that is not time critical. Requirements on the interface were shown that cannot be expressed in WSDL. It was proposed to formulate these in the specification technique SXQT based on first order logic. The automatic validation can then be used to ensure that non-conforming messages are rejected and thus cannot compromise the fieldbus system.

Acknowledgments

This work was supported in part by the Hochschuljubiläumsstiftung der Stadt Wien via project CoMa (H-965/2002).

References

- [1] W. Elmenreich, S. Pitzek: "Smart Transducers - Principles, Communications, and Configuration" In: Proceedings of the 7th IEEE International Conference on Intelligent Engineering Systems (INES), p.510 – 515, March 2003.
- [2] M. Horstmann, M. Kirtland: DCOM architecture. White paper. Microsoft Corp., Redmond, July 1997. Available at:
http://msdn.microsoft.com/library/en-us/dndcom/html/msdn_dcomarch.asp.
- [3] CENELEC: General purpose field communication system. Standard {EN} 50170, Vol. 2/3 (Profibus). European Committee for Electrotechnical Standardization. December 1996.
- [4] Smart transducers interface final adopted specification. Object Management Group (OMG), August 2002. Available at <http://www.omg.org> as document ptc/2002-10-02.
- [5] H. Kopetz: Software engineering for real-time: A roadmap. In: Proceedings of the IEEE Software Engineering Conference, Limerick, Ireland, June 2000.
- [6] N. Mitra: SOAP Version 1.2 Part 0: Primer. W3C Proposed Recommendation. World Wide Web Consortium (W3C), May 2003. Available at:
<http://www.w3.org/TR/2003/PR-soap12-part0-20030507/>.
- [7] R. Chinnici, et al.: Web Services Description Language (WSDL) Version 1.2. W3C Working Draft. World Wide Web Consortium (W3C), March 2003. Available at:
<http://www.w3.org/TR/2003/WD-wsdl12-20030303/>.
- [8] D. C. Fallside: XML Schema Part 0: Primer. W3C Recommendation. World Wide Web Consortium (W3C), May 2001. Available at:
<http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/>.
- [9] C.A.R. Hoare: Communicating Sequential Processes. Prentice-Hall International, London, UK, 1985.
- [10] S. Boag, et al.: XQuery 1.0: An XML Query Language. W3C Working Draft. World Wide Web Consortium (W3C), May 2003. Available at:
<http://www.w3.org/TR/2003/WD-xquery-20030502/>.