Application specific vs. standard Web service interfaces for the vertical integration of fieldbus systems

Marcus Venzke[1], Christoph Weyer[1], and Volker Turau[1]

[1]Department of Telematics,
Hamburg University of Technology, Germany
{venzke,c.weyer,turau}@tu-harburg.de

Abstract — *The paper compares two approaches for developing Web service interfaces for the vertical integration of TTP/A fieldbus systems. High-level abstractions are provided by application specific interfaces, generated from metadata describing fieldbus systems. In contrast standardised interfaces such as OPC XML DA only allow lower levels of abstractions. But these enable accessing the fieldbus system from a broad range of standard clients, while high-level abstractions reflecting the application programmer's view on the system facilitate the development of more specific clients and workflows.*

1 Introduction

Enterprises increasingly aim for the integration of their information systems to satisfy market requirements for flexibility, speed and cost reductions. Even automation systems are integrated to enable the vertical integration from high-level management systems down to production systems in the factory shop floor and for supporting condition monitoring, fault diagnosis and predictive maintenance. Their integration raises specific obstacles due to the mismatch between their capabilities as autonomous, real-time oriented automation systems and the expressiveness in high-level information systems. In the past this has often led to software being complex to master and verify.

Web services are a suitable technology for the vertical integration of automation systems, allowing loose coupling of autonomous systems. Justification and discussion of requirements for Web service interfaces are given in [1]. These include security issues and handling concurrent access to provided services. Interfaces need to be coarse grained, because of the high latency of Web service calls. A major requirement is to provide high-level abstractions to ease the integration.

This paper presents and compares two fundamentally different approaches of how to engineer such interfaces: application specific and standardised Web service interfaces. Having TTP/A fieldbus systems in mind one example is discussed for each approach. Section 2 reviews the integration primitives available in TTP/A fieldbus systems. After describing

the standardised Web service interface *OPC XML-DA* in section 3, section 4 presents how to map these to TTP/A's primitives. The mapping of automatically generated, application specific interfaces is discussed in section 5. Both approaches are compared in section 6, before section 7 concludes.

2 TTP/A fieldbus systems

This paper primarily considers TTP/A fieldbus systems [2] consisting of nodes ("smart transducers") connected by a bus allowing real-time communication. Each node contains a micro-controller with sensors or actuators. As shown in Figure 1, a gateway enables external access. It may be used by a Web service provider to connect to the system.

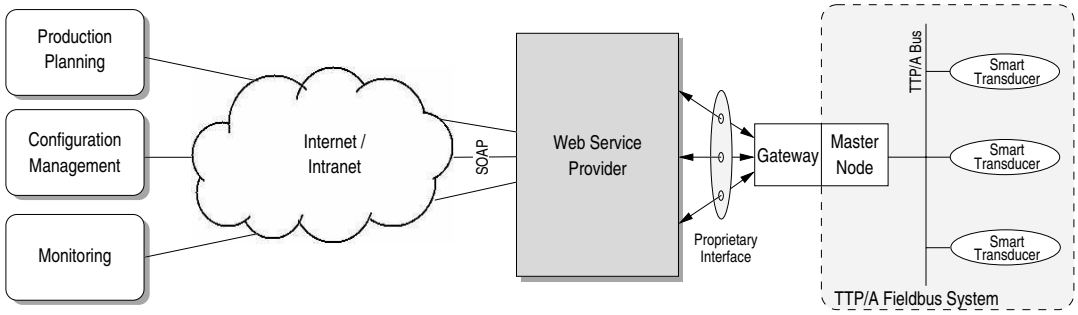

Figure 1: Basic architecture for vertical integration

According to [3] access should be supported with three interfaces viewing nodes from three different perspectives. The *real-time services interface (RS)* provides time sensitive information generally used for control purposes. The *diagnostic and management interface (DM)* enables monitoring and administration. Nodes are configured using the *configuration and planning interface (CP)*

A key feature of TTP/A fieldbus systems is the concept of an *Interface File System (IFS)*, a conceptual model for addressing data distributed across the nodes. Addressing is based on a static four-level hierarchy in which fieldbus systems (clusters) contain nodes, organised in files consisting of 4 bytes records. Entities on each level are identified with a one byte number allowing to uniquely name each record with a four bytes address.

IFS supports three file operations on records: *read*, *write*, and *execute*. All operations have the record's four bytes address as parameter. The operations *read* and *write* atomically read or write a single named record. When calling the operation execute, the record's address denotes a function to be performed on a node. If the function requires parameters these have to be written to other records in advance.

Metadata is added to a TTP/A fieldbus system to allow generic software to interact with it. This enables application software (e.g. management tools) to adopt itself to different or changing fieldbus systems. The paper utilises XML-based metadata format introduced in [4] similar to the format from [5]. It describes a fieldbus system with one cluster description and one or many node descriptions.

A *node description* specifies the characteristics of a specific type of nodes. It is published by the node manufacturer and includes descriptions of existing files, records and operations. Logical variables are assigned logical names and are mapped to a single or

several IFS records. Supported operations are defined including their names and parameters. The *cluster description* describes a fieldbus system as a whole. It is created for a specific cluster and used to configure it. Constituent nodes are named and described as well as the communication amongst them. Their types are defined by referencing node descriptions.

3 Standard interface: OPC XML-DA

The OPC XML-DA [6] is a standard Web service interface for reading and writing data from and to plant fbor automation systems. It is defined by the OPC Foundation, an industry consortium of over 300 members including the major vendors. The majority of interfaces defined by the consortium are based on Mircosoft's *Component Object Model (COM)*, mainly available for MS Windows platforms. XML-DA is their first interface for Web services allowing interoperability with a broader range of platforms.

OPC XML-DA's data model is based on typed data items (*OPC items*) that are named and organised in a hierarchy. Each item stores a single value. It is identified with the combination of the two strings: *item path* and *item name*, the item path identifying a namespace in which the item name is unique. A set of dynamically retrievable *properties* is associated with every item containing its metadata including a human readable description, access rights, a time stamp, change rate, engineering unit and data type. Possible data types are simple (e.g. string, integer or double), enumerations or arrays.

Operations for accessing item values are *Read* and *Write*. Both allow accessing several items with a single call. Each item's path and name is contained in requests to these operations. To optimise periodic reads of the same set of items a subscription mechanism is provided. The set of items is subscribed by calling the operation *Subscribe* and then periodically polled using *SubscriptionPolledRefresh*.

The OPC operations *Browse* and *GetProperties* are used to query which OPC items are available and values of their properties. *Browse* allows querying an OPC item's immediate successors including filtering and can return property values of the items found. Property values can also be retrieved with *GetProperties*. The operation *GetStatus* is used to retrieve the status of the OPC server.

4 Mapping OPC XML-DA to IFS

Providing OPC Web services for TTP/A fieldbus systems requires an OPC XML-DA abstraction for IFS. IFS's fixed four-level addressing hierarchy needs to be mapped into the address space of OPC. The three interfaces RS, DM and CP have to be implemented with the sole interface of OPC XML-DA. Moreover the different semantics of their operations have to be bridged. Finally the metadata describing the fieldbus system must be represented as OPC properties.

While IFS's addressing hierarchy is fixed to clusters, nodes, files, and records, OPC has namespaces allowing arbitrary hierarchies. This enables a direct mapping. Clusters, nodes, files, and records are the items constituting the hierarchy. Clusters contain nodes as children, having files as children. Records could be used as leafs having four byte values. All items could be named with IFS's numbering scheme.

However metadata describing the fieldbus system allows improving this hierarchy. The

node description defines logical variables including types and how these are composed from records. This enables typed variables as leaf items of OPC's hierarchy, abstracting from individual storage spaces and adding semantics. In addition the cluster and node descriptions define names for clusters, nodes, files and logical variables. Naming OPC items with these increases the understandability of item paths and item names used in OPC XML-DA.

Having only one interface in OPC XML-DA raises the issue of how to represent IFS's three interfaces RS, DM, and CP. The reason for introducing three interfaces was to reduce the complexity of an individual interface by only supporting primitives required for a specific view on a fieldbus node. The interfaces RS, DM, and CP represent the three views of accessing real-time data, for diagnostic and maintenance, and for configuration and planning respectively as proposed in [3]. The same views should be supported with OPC XML-DA's single interfaces as part of OPC's hierarchy. We propose to represent each view as OPC item as children of a cluster, having descendants of only those nodes and logical variables which are accessible in that view.

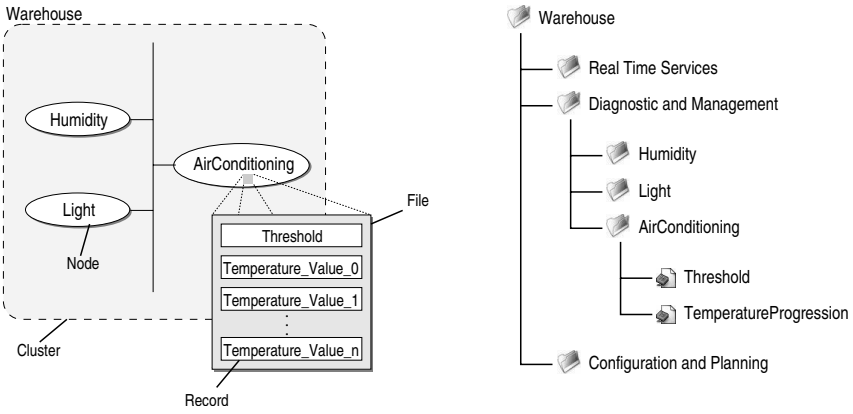

Figure 2: Mapping strucure of IFS to OPC Items

Figure 2 depicts a mapping of an example IFS structure to the OPC items hierarchy. On the left side a simple fieldbus system with a single cluster (*Warehouse*) is shown. The cluster contains three nodes *Humidity*, *Light*, and *AirConditioning*. The latter contains a file with records regarding the temperature threshold and a set of temperature values describing the progression in time. The right side of figure 2 shows the OPC item hierarchy representing the OPC view upon the fieldbus system. As proposed the cluster contains the three different interfaces for accessing the nodes. Records are mapped directly on OPC items. The set of records describing the temperature progression is merged into one OPC item with the data type array of integer.

Future approaches may completely hide the IFS hierarchy, providing an OPC hierarchy based on the item's meaning for the application. To construct this hierarchy metadata is required classifying logical variables from this point of view. For example, all variables representing measured temperature values may be assigned a specific identifier. Unfortunately such classification is not available in current metadata formats for TTP/A fieldbus systems.

Bridging the semantics of the operations from IFS to the OPC XML-DA is obvious

to some extent but raises issues for IFS's operation *execute* and for metadata. OPC's operation *GetStatus* returning the current status of the server does not need to be mapped at all, but is directly implemented by the Web service.

OPC's operations *Read* and *Write* are mapped to the IFS primitives with the same name and similar semantics. A single OPC call may be mapped to several calls in IFS for two reasons. Firstly one logical variable in OPC may be mapped to several IFS records, as described above. Secondly OPC allows reading or writing many values with one call to achieve a coarse granularity. This is required for scalability in high latency networks such as the Internet [1].

Reading and writing over the RS interface requires considering real-time, which cannot be achieved over networks based on Internet technology and thus with OPC XML-DA. To overcome this issue OPC provides a timestamp with every value that is read giving the time when the value was valid. On the RS interface the timestamp should be set to the point in time when the value was valid in the TTP/A fieldbus node. Determining this point in time is not possible if the DM or CP interface is used, because the protocols used for these interfaces on the TTP/A fieldbus do not guaranty real-time. The time can only be approximated by the Web service implementation.

OPC's mechanisms for subscriptions can directly be implemented in the same manner as the operation *Read*. But more elaborate techniques allow optimisations such as preloading and caching of data [1].

The IFS operation *execute* raises the issue, that it does not exist in OPC. Hence it needs to be mapped to *Write* or *Read*. *Write* is preferred because there is general agreement that *Read* does not cause changes in the system. In IFS the same record may be allowed to be executed and written. If so ambiguity should be eliminated by defining two logical variables for *write* and *execute*, even though only one is defined in the cluster description. Both should be differentiated by name. Since *execute* does not allow a parameter the value required for OPC's operation *Write* has to be ignored.

The operations *Browse* and *GetProperties* primarily return metadata from the cluster and node descriptions, specifying the hierarchy and the logical variables properties. OPC properties described in the metadata include the item's data type, minimum and maximum value, engineering unit (e.g. meter), change rate of the value, access rights (readable and/or writeable), and a human readable description. Three other properties require retrieving data from the fieldbus system as with the operation*Read*. These contain the item's value, the related timestamp, and the quality marker. The latter describes the value's accuracy to be good, bad, unknown, or inaccessible due to an error. It needs to be mapped from the numeric value used in IFS for the same purpose and from error states.

Table 1 summarises the proposed mapping of OPC XML-DA operations to IFS operations and the usage of properties from the metadata describing the cluster.

In order to prove the mapping of OPC XML-DA for TTP/A fieldbus systems a prototypical implementation exists [7] using the XML-DA Rapid Server Toolkit (XDARap) from Advosol [8]. XDARap is a .NET Web service that implements the OPC Foundations OPC XML-DA specification version 1.0.1. Figure 3 shows the general structure of the OPC XML-DA Web service. The generic part manages the OPC XML-DA client interface. The concurrent update of the data values in the generic part and the customisation part is done by the update thread. A cache is used to store values for the subscription

OPC XML-DA	IFS	Used Metadata
GetStatus	—	—
Read	read	*dataType property*
Write	write / execute	*dataType property*
Subscribe	—	*scanRate property*
SubscriptionPolledRefresh	read	*dataType property*
SubscribtionCancel	—	—
Browse	read	specified properties
GetProperties	read	specified properties

Table 1: Mapping of OPC XML-DA operations

mechanism. All items are maintained in a hierarchical address space by the generic part.

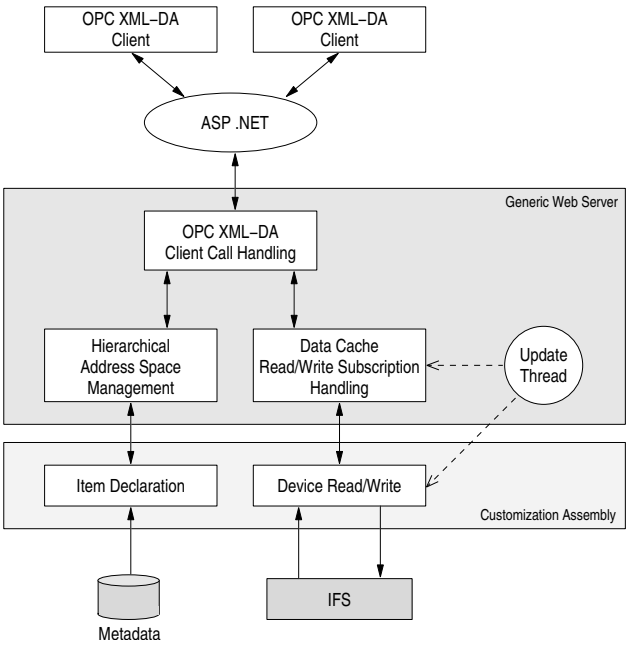

Figure 3: Architecture of OPC Web service implementation

The customisation part is implemented as a .NET assembly using C#. This part handles all device specific activities. On startup the metadata is used to create the item hierarchy and to configure the item properties. During runtime the read and write operations are handled by a device specific adaptor. A IFS simulation developed in [9] is used to verify the mapping and the implementation.

5 Application specific interfaces

As an alternative to OPC XML-DA we have proposed to generate cluster specific Web service interfaces based on a cluster's metadata [4]. The three interfaces RS, DM, and CP are retained as individual Web service interfaces having different operations as required for the specific purpose. For configuring a cluster its cluster description is given to the Web service via the CP interface. This determines logical names and types to be used in

the RS and DM interface and creates logical operations.

Figure 4 depicts the conceptual architecture for implementing the three interfaces. Generic software components are combined with cluster specific components. The latter support naming and typing of data and implement cluster specific operations. Descriptions of the operations are included in the WSDL document, which describes all three Web service interfaces. Generic software components as well as the WSDL document are generated by the so-called *configuration processor* when configuring the cluster. It is based the cluster description including the referenced node descriptions.

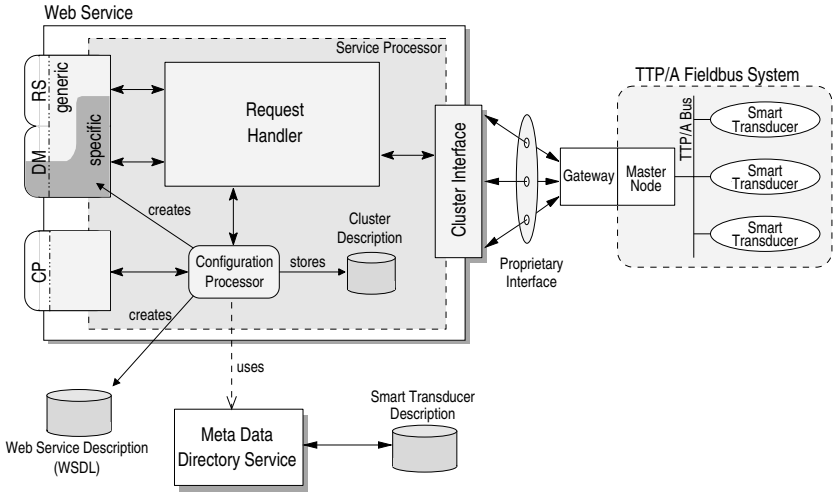

Figure 4: Architecture for implementing application specific interfaces

The Generation process is activated when the cluster is configured by calling the operation *Configure* over the CP interface. The cluster description is passed as a parameter in the form of an XML document. Using this as input the configuration processor not only generates the cluster specific components but also configures the cluster and saves the description for later retrieval. The cluster description is returned by the operation *ReadConfiguration* again as XML document. Since the configuration needs to be accessible from all views this operation is available in all three interfaces.

The RS interface allows retrieving time sensitive data periodically exchanged on the TTP/A bus. The semantics of the operations is similar to OPC XML-DA. Logical names and types are extracted from the metadata. Logical names are used to identify data retrieved with the operation *Read*. Returned values are typed and may originate from multiple IFS records. A time stamp attached to each value provides real-time semantics. To achieve a coarse granularity many such values can be retrieved with a single call by allowing lists of logical names and values as parameter. Also a mechanism for subscription is provided with operations similar to OPC XML-DA.

The DM interface supports maintenance tasks such as monitoring, fault diagnosis and parameterisation. This requires reading and writing values from and to fieldbus nodes and executing logical operations. Operations for reading and writing are again similar to OPC XML-DA. The operation *Read* allows reading data as in the RS interface, but without attaching time stamps to values. *Write* is provided for writing sets of logical, typed values into the fieldbus system.

For executing logical operations in fieldbus nodes the DM interface is enhanced during the configuration. A node's logical operations are described in its node description that is referenced from the cluster description. The configuration processor uses both types of metadata to add the logical operations to the DM interface as Web service operations. When an operation is called parameters are automatically written into the appropriate records, the IFS operation execute is called and return parameters are read.

6 Comparison

Web service access to TTP/A fieldbus systems can be achieved with both application specific interfaces and the standard interface OPC XML-DA. Both approaches provide a course granularity required for Web services and a higher abstraction than IFS. But differences in their semantics and standardisation makes them suitable in different application scenarios.

The two approaches vary in how they represent configurations and capabilities of individual fieldbus systems. OPC XML-DA preserves a single interface with the same semantics over a broad range of automation systems including TTP/A fieldbus systems. Differences are reflected in the hierarchy of its namespace, existing items, names, and properties. With application specific interfaces these are represented by adding operations reflecting the fieldbus system's particular capabilities. The three views RS, DM, and CP are retained as individual Web service interfaces making operations also view specific.

The major advantage of having OPC XML-DA as a fixed, standardised interface is the enabling of standard clients. Applications only need to support a single interface for accessing a broad range of automation systems. It is adapted to a specific system by configuring item paths and item names of logical variables that shall be read from or written to. Industry applications already support XML-DA. An example is Siemens's process and production visualisation tool *SIMATIC Windows Control Center* [10]. SAP R/3 provides the *OPC-DA Connector* [11] supporting the OPC DA interface that can be adapted to OPC XML-DA with standard proxies.

Accessing a fieldbus system over application specific interfaces requires an application to have generic support for calling Web services. Generally a proxy is generated from the interface's WSDL document, representing the Web service operations as local functions or methods. Program code then needs to be implemented to integrate the proxy into the application. Some industry applications support this generic kind of calling Web Services, SAP R/3 is again an example [12].

Even though supporting OPC XML-DA is less complex for standard clients, the high-level abstractions provided by application specific interfaces are more suitable for developing more specific clients. Applications specific abstractions are constituted to reflect the application programmer's view on a specific fieldbus system. This makes the development more straightforward and less error-prone.

An example are logical functions with parameters that can be called in fieldbus nodes. In an application specific interface a function is represented as Web service operation that is provided for development as local function by the proxy. The function can be called directly providing parameters and returning results. Calling the same function over an OPC XML-DA interface requires three steps. The operation *Write* has to be called for setting all logical variables containing parameters. The function is then executed by

calling *Write* for the logical variable representing the function. Finally *Read* needs to be called for querying logical variables for the results. Performing the three steps requires an application programmer to know details about which variables represent parameters, results and functions and increases the effort of error handling.

Similarly configuring a TTP/A fieldbus system requires one call with an application specific interface or setting many logical variables with OPC XML-DA. A single call to the operation *Configure*described in section 5 configures the fieldbus system having the XML encoded cluster description as parameter. From the cluster description the operation determines appropriate values for the configuration relevant records in all fieldbus nodes. With OPC XML-DA all equivalent logical variables have to be set with the operation *Write*. It is the responsibility of the application programmer to know which variables have to be set to which values.

When integrating the Web service into workflow systems, application specific primitives also increase the readability of workflow descriptions. [13] sees workflows as an important approach for the vertical integration of automation systems. Languages such as BPEL4WS (Business Process Execution Language for Web Services) are used to describe Web service based workflows as processes consisting of activities that are calls to Web service operations. Thus the operations directly become the primitives for describing workflows and should be meaningful from a workflow perspective which is similar to the application programmer's view. As example an activity in an error handling workflow of a shop floor might be to reset a fieldbus system. An application specific interface may provide an operation *Reset* for that purpose, also becoming the primitive in the workflow description. In contrast with OPC XML-DA the primitive would be the operation *Write* having the name and path of a logical variable as parameter, that represents the function of resetting the system. Having the latter primitive in a workflow description obviously provides less readability.

Another advantage of application specific interfaces is the ability of checks at the client's compile time. Proxies are generated from WSDL documents containing information about operations, parameters and types. This allows type checking when compiling and linking proxies with the application. With OPC XML-DA's operations *Browse* and *GetProperties* items and types are retrieved dynamically and can thus only be checked at run time.

Application specific interfaces also allow a more flexible authorisation scheme. Authorisation in OPC XML-DA may disallow calling the operations *Write*, *Read*, or querying metadata with *Browse* or *GetProperties*. A more fine grained scheme would require putting rights on individual OPC items. Similar options exist with application specific interfaces. But putting access rights on the three interfaces and the application specific Web service operations provides further options for a simple, yet flexible authorisation scheme.

7 Conclusion

Both approaches - application specific and standardised Web service interfaces - have their advantages justifying their utilisation in specific fields. Having a fixed, standardised interface such as OPC XML-DA for a broad range of applications, enables standard clients to access many systems by simply implementing that interface. This is a signifi-

cant advantage, even though the required common abstraction cannot reflect the application programmer's view on a specific automation system. Application specific interfaces support that view, thus facilitating the development of clients and workflows implemented for that application. But integrating these into standard clients requires generic Web service support and implementation effort for individual systems. Thus standardised Web service interfaces should be chosen for automation systems mainly accessed by standard clients, while application specific interfaces should be used for systems to be integrated into Web service based workflows and specifically implemented environments.

References

[1] V. Turau, M. Venzke, and C. Weyer. Vertical Integration of TTP/A Fielbus Systems Using Web Services. In *Proceedings of First International Conference on Informatics in Control, Automation and Robotics (ICINCO 2004)*, Setubal, Portugal, August 2004. IEEE Computer Society Press.

[2] H. Kopetz, M. Holzmann, and W. Elmenreich. A Universal Smart Transducer Interface: TTP/A. In *Proceedings of the Third IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC'00)*, March 2000.

[3] H. Kopetz. The Three Interfaces of a Smart Transducer. In *Proceedings of the Forth IFAC International Conference on Fieldbus Systems and their Applications (FeT'01)*, Nancy, France, 15th-16th November 2001.

[4] V. Turau, M. Venzke, and C. Weyer. A Web Service for TTP/A Fieldbus Systems based on Meta-Data. In *Proceedings of the Second Workshop on Intelligent Solutions in Embedded Systems (WISES'04)*, pages 159–168, Graz University of Technology, Austria, 25th June 2004.

[5] S. Pitzek. Description Mechanisms Supporting the Configuration and Management of TTP/A Fieldbus Systems. Master's thesis, Vienna University of Technology, Austria, August 2002.

[6] OPC. *OPC XML Data Access Specification* Openness Productivity and Connectivity Foundation (OPC), November 2003. Version 3.00.

[7] S. Liu. OPC XML DA for accessing IFS. Master's thesis, Hamburg University of Technology, Germany, January 2005.

[8] Advosol. XML DA Rapid Server Toolkit, 2004. http://www.advosol.com/pc-12-9-xml-da-rapid-server-toolkit.aspx

[9] S. Bartkus. Integration of Fieldbus Systems into Enterprise Applications based on Meta Data. Master's thesis, Hamburg University of Technology, Germany, September 2004.

[10] Siemens. SIMATIC WinCC - Prozessvisualisierung und Plattform für IT & Business Integration, 2004. http://www.siemens.de/wincc

[11] SAP. SAP OPC Data Access (SAP ODA), 2000. http://www.gefanucautomation.com/downloads/special/features/sapoda_paper.doc

[12] A. Schneider-Neureither, editor. *The ABAP Developer's Guide to Java*. SAP Press, 2004.

[13] A. P. Kalogeras, J. Gialelis, C. Alexakos, M. Georgoudakis, and S. Koubias. Vertical Integration of Enterprise Industrial Systems Utilizing Web Services. In *Proceedings of the Fifth IEEE International Workshop on Factory Communication Systems (WFCS'04)*, pages 187–192, September 2004.