A COMMON MODEL FOR XML DESCRIPTIONS IN AUTOMATION

Martin Wollschlaeger*, Henry Kulzer, Daniel Nübling***, Peter Wenzel*****

** Industrial Communications, Department of Computer Science,*
TU Dresden, D-01062 Dresden, Germany
*** SIEMENS AG, Systemtechnik SIMATIC, IT Based Automation*
A&D AS RD DHN 3, PO Box 4848, D-90327 Nuremberg, Germany
**** PROFIBUS International, Haid-und-Neu-Str. 7, D-76131 Karlsruhe, Germany*

Abstract: The use of XML as a description language has become state of the art within the automation and control domain. Use cases and application scenarios are manifold, resulting in heterogeneous XML document structures. However, a common basic model for XML Applications is still lacking. This paper describes requirements and principle structures for such a common model, which has been developed for use within PROFIBUS and PROFINET systems.

Keywords: Automation, Description, XML, Internet, Software Tool.

1. MOTIVATION

An automation and control system consists of several networked components; each with its dedicated functionality and with a specific information set. The communication between these networked devices is performed via industrial communication systems like fieldbuses (IEC 2002) or industrial Ethernet solutions (PI 2003). Besides the data exchange at runtime, these components provide an information set containing additional data like configuration information, maintenance information, documentation and so on. Several software tools are used to control and to adapt the functionality and the information sets of the components, depending on application requirements. These requirements are determined by the automation and control application. Depending on their functionality and on their position within the life cycle of an automation system, the software tools handle the components within different contexts. The scenario described above implies extensive, context-depending exchange and interpretation of data.

The environment for software tools and for components is heterogeneous (see Wollschlaeger *et al*., 2003). The interfaces for data exchange or interpretation usually are also highly heterogeneous and are often of proprietary nature. Some suppliers have standardized the interfaces for a data exchange between their own products or have implemented specific converters to support systems of other suppliers. These 'point-to-point' converters are mostly implemented as proprietary solutions, are equipped with a binary data format, are not easy to use and form a close relation between the current releases of applications, which may fail after an update of one or both releases. Furthermore, there are a lot of export interfaces using an ASCII or an ASCII like (e.g. CSV) data format.

During the last years, a large number of specific data exchange formats have been defined, partly depending on a model like STEP (ISO 2004) or like the Device Model according to ISO 15745 (ISO 2003, FDCML 2002), and partly as private specifications. In modern solutions, XML (W3C Consortium, 2000a) is used as a bidirectional data format. In the moment, XML formats are mostly defined without coordination to other products, not to mention other suppliers. Data are being described by means of XML, but their designation and structure, as well as their semantics differ substantially. This leads to

incompatible data formats, wasting the benefits of XML (Fig. I).

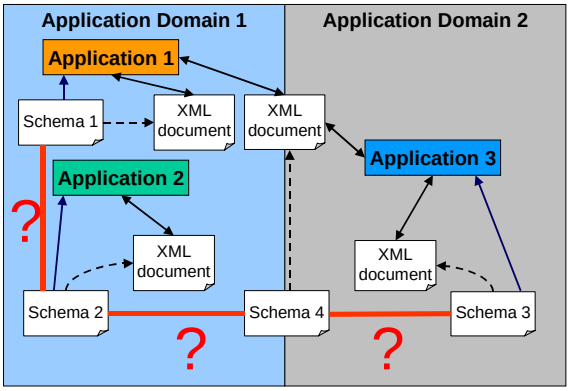

Fig. I. Current situation in application of XML in the automation domain.

2. USE CASES AND REQUIREMENTS

Some of the problems not solved by or even just introduced by XML are missing restrictions for document structure, no restrictions for schema structure, no common model. Furthermore, XML is a "moving target", requiring to critically evaluate new specifications, technologies etc. All this leads to inhomogeneous documents.

To provide solutions for the problems and limitations described above, the most relevant use cases for application of XML in automation systems have to be elaborated in order to define requirements for using XML and to build up a more generic model for common use (Wollschlaeger *et al.*, 2004; PI 2004).

2.1 Use Cases

For historical reasons, one of the main use cases is *Device Description.*
Device Descriptions based on a Schema are well suitable for generic tools, code generation, interpretation and transformation. XML files can capture all facets of device interfaces and features, including nested information without any restriction to nesting. Device descriptions are a key technique to integrate devices into different automation environments.

A second important use case is *Data Exchange.*
This use case can be sub-divided into the following ones:

- Data Migration due to an update of the Application Software

The user wants to re-use data with a newer version of application software. Upon an application software update, it is very important for the user to have a migration facility for his data. Up to now, conversion tools and bridge programs had to be designed for each application. With a standardized export format based on XML, there is no or only a little effort required, if a few directives are observed.

- Data Exchange between applications and manufacturers

Today, a proprietary interface is used by applications in order to exchange data like project structures or device information. The proprietary interface also delivers 'internal' information of the exporting application, which has to be imported and analyzed by the other applications. This causes more effort and a very close coupling of the applications (like the need to update the import functions after a modification of an export function inside another application).

- Context-specific interpretation of Data

Some applications import data produced by another application. The importing application has to be able to identify data it can interpret, and has to separate this data from unknown information, which has to be handled as 'black box'.

- Runtime Data Exchange

Applications and devices can exchange data at runtime with remote procedure calls (RPC) or web services with the benefit of a common interchange format.

Another use case is a *comparison of project data*
A complete export of project information in an easily interpretable data format makes it possible to identify differences and inconsistencies in various versions of the same project. A comparison shall take into account a specific context, and it shall be possible at different levels of project data.

Controlling functionality of applications is also a typical use case.

- Application-specific parameterisation of software tools

The functionality of software tools may be derived from XML documents or schemas. An XML file or schema can be used to describe user interface components, or functionality of applications. These descriptions can be interpreted by the tools at runtime. Flexible tools may be created this way, allowing their extension or adaptation without re-programming. Furthermore, a unique behaviour of a given functionality can be achieved.

- Generic Tool

Along with strategic comments and annotations schemas can be evaluated by a tool to set up intuitive user interfaces or tools. Thus a generic and only schema driven document editor could be made which is able to leave a user in his own (business) world, not forcing him down to know about XML schema. Without such an editor, owners of different schemas probably have to have individual editor tools to generate instances of their schemas (documents), unless these instances will be of machine generated nature.

Finally, *administrative information* can be described in XML.
Administrative information and Meta data like versions of documents, author information, and so on shall be defined in the same way throughout different application domains.

2.2 Requirements

Based on the use cases, some basic requirements have been defined (see PI 2004). The most important are listed below:

- Common, flexible content model
- Identification of content
- Avoidance of name conflicts
- Extensibility
- Reference mechanism for content and hierarchies
- References to external content
- Import/Export of Subsets
- Export of subsets with ‘open’ references
- National language support
- Versioning of schemas, documents and contents
- Time stamp on various levels:
- Splitting into multiple files
- Tool specific content identification
- Object centric description
- Data compression for XML documents
- Know-How Protection, content encryption

The requirements have to be taken into account when XML schemas and applications have to be developed.

PROFIBUS (IEC 2002) and PROFINET (PI 2003) are widely used industrial communication systems. For applications in such environments, the guideline XML@PROFIBUS (PI 2004) has been developed containing a common model, which shall be used for schema definitions and XML document structuring. Furthermore, rules for namespaces and versioning have been defined, a workflow is described and a supporting infrastructure has been set up. Based on this guideline, building blocks of re-usable schema definitions and instance files have been defined, which will be extended on demand. As a result of this process, the relations between XML instance files and schemas change, as it is displayed in Fig. II.

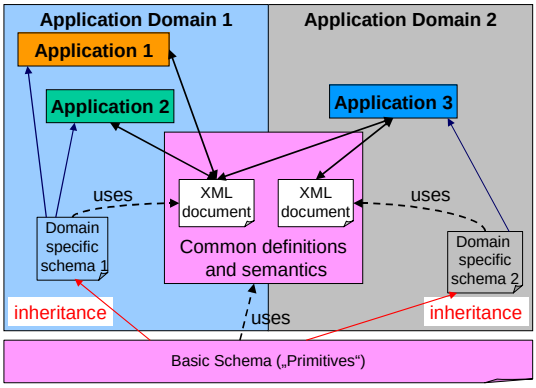

Fig. II. Intended use of XML based on common definitions.

3. COMMON MODEL

The common model is based on an object structure (Fig. III). An object is used as an abstract construct, where application-specific objects can be derived from. As a required attribute, it contains an ID. It may be further characterized by a type name. A readable name may be assigned to the object. An object may have a version, describing the version of content the object contains. Objects can be organized hierarchically, depending on application requirements. Thus, an object contains all information necessary for uniform identification, a major prerequisite for a common model.

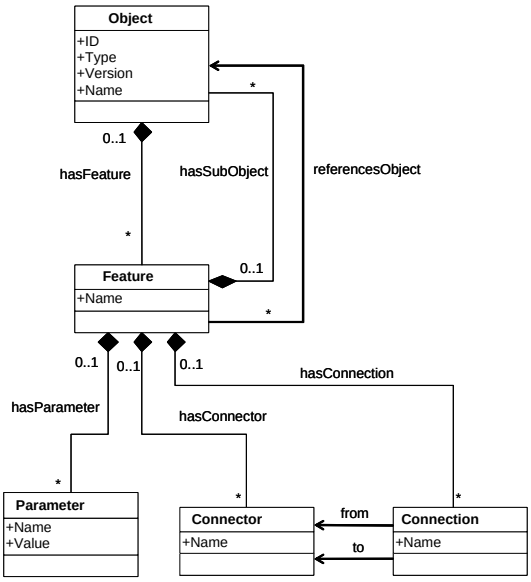

Fig. III. The object as the basic construct of a common model.

Besides the general object identifier, application specific identifiers are supported as well, because applications usually require a relation between their specific objects and their exported objects. By this method software tools within a tool chain may apply their own namespaces, naming conventions and their specific identification rules without conflicting with each other. In addition, a tool may add specific information, and may retrieve this information later on; regardless of the other tools of in a chain.

Since an object contains further details in its content, a structure has to be developed for organizing the content. Instead of simply adding content elements as tags underneath the base object tag, a more generic structure has been defined. The content is mapped to named parameters. The parameters are grouped by features. Such a named feature shall reflect a specific logical group of parameters. As a rule of thumb, features should be disjunctive to each other, meaning that they should contain context-depending parameters without an overlapping. For example, any display-related information of an object should go into one feature, while any access-related information should go to another one. The benefit is obvious – further features can be added flexibly, without redesigning the object.

Features can reference other objects. Basic reference mechanisms have been defined, applicable for local references, for relative references and for references to objects or features in external files. Object identification as described above is used for specifying the target. Well-known, widely accepted standards like XPath (see W3C Consortium 1999) are used. Thus, an intrinsic, flexible referencing methodology is defined, which is a prerequisite for a flexible distribution of objects and features to multiple files.

For several reasons, it may be necessary to encrypt the content of an object. An insertion point for stan-

dard technologies within the XML domain is embedded in the object.

If, besides a logical association, a relation between objects has specific parameters, it can be explicitly modelled using connectors and connections. Within the base object, abstract connectors and connections have been defined, where specific ones shall be derived from.

While the object is a basic construct for modelling data, a document is the basic element for any XML instance file. The basic document type contains Meta information describing user, tool, and timestamp of creation or modification. Furthermore, attributes necessary for referencing satellite files (distribution of objects, or language-depending definitions). Finally, a document may be entirely encrypted (see W3C Consortium 2001b). An adequate insertion point for a signature is embedded.

The document contains objects defined using the object model described above. Whether a derived object is allowed to be inserted into a document or not is controlled by the substitution group mechanism. When specifying objects, an adequate definition of substitution groups can be used to control the structure of the document tree. Although – compared to the usual approach of modelling tag hierarchies directly – this method requires an extra effort in the design phase of a schema, but it provides an outstanding flexibility which is essential in many use cases.

XML documents have an implicit hierarchy, reflected by the tag structure. However, it is often required to interpret the data in a way, where the implicit hierarchy cannot be used. Therefore it is possible to model hierarchies in an explicit way, depending on the context. This way more than one hierarchy can be integrated into the same document, without having to change its internal structure. These explicit hierarchies are described using the reference methods described above.

Another prerequisite mentioned above is support for multiple languages. This is provided by using language-depending parameters, which have a standard "xml:lang" attribute to indicate the language. Since it's frequently desired to separate language-depending text (e.g. labels) form the main document, a uniform way of distributing text to satellite files or to one specific tag within the main document has been defined.

4. SCHEMAS AND NAMESPACES

The detailed, application-specific definition of XML documents shall be done using XML schema rules according to (W3C Consortium 2001a). However, the common model described above shall be considered. In order to do this, a basic schema ("Primitives") has been defined containing the elements of the model, and some additional building blocks. This schema shall be referred to within own schema definitions.

The reference to the Primitives is done by means of standard XML and schema mechanisms. This allows either including, importing or redefining of the Primitives. An adequate namespace prefix has to be used in the respective statement.

In order to support the idea of a common model, the specific schemas have to fulfil some requirements. First of all, entities defined by a schema shall be described as an object derived from the object in the Primitives. This is no restriction to generality, but a prerequisite for seamlessly applying common identification methods, references, multiple languages etc. At second, the content of a schema shall follow a structure as shown in Fig. IV. It shall be considered, if there are adequate definitions already existing which can be re-used. If not, an entity shall be defined keeping in mind its ability to be re-used. It is desired practice to define domain-specific schemas, which contain all entities relevant for an application domain, while an application-specific schema shall only contain entities specific for the application. In a domain-specific schema, the objects of the Primitives are refined to automation specific objects. For example, shared data types may be defined, which can be used by various tools and build a common base. Although the assignment described above should be done during initial schema development, schema evolution with versioning and namespaces allows future adjustments.

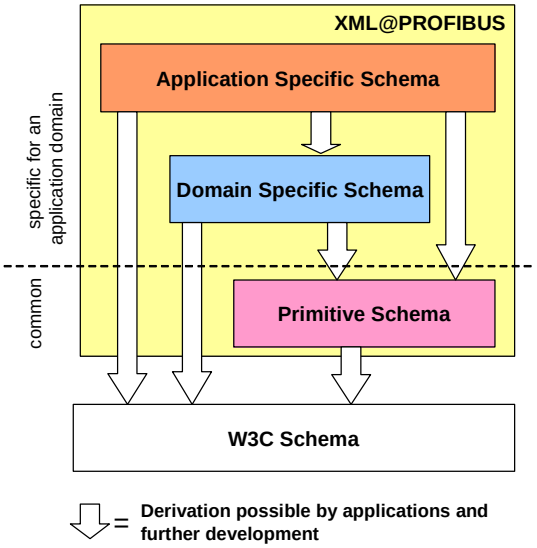

Fig. IV. Architectural overview of schema dependencies.

By qualifying the namespaces, instance documents may comprise of elements defined in each of the underlying schemas. For namespace names, some basic rules have been defined. As usual, the namespace name is a string corresponding to a URI. This is done for two main reasons – responsibility and support.

Since a URI contains a web address of an organization, only this organization can bear responsibility for

the URI and for the namespace. It is also a task of that organization to specify further details of the namespaces in a consistent way. The preferred method is a general subject followed by year and month and by application areas (perhaps with further qualifying sub-elements).

The support aspect is fulfilled by having the namespace URI pointing to a HTML page containing at least a short description of the schema, a link to download the schema file, a link to the schema documentation and an exemplary instance file. Thus, any developer of schema files can gain optimal support for his work. Fig. V shows a screenshot of such a HTML page available at PROFIBUS International.

Fig. V. Screenshot of a support page.

5. EXTENSIBILITY

The systems within the automation domain are rather complex. Furthermore, they are often used as embedded or underlying structures of huge systems, like for example enterprise control systems. Considering the different phases of the life cycle of automation systems, additional views to a system as a whole and to its components are introduced.

An application of XML in this context requires structures and methods for an easy adaptation to the required targets. This can be achieved in several ways. The most suitable one is the combination of elements from different namespaces within a schema. By qualifying the namespaces according to standard XML rules (as mentioned above) specific schemas can be created. However, this requires a kind of “early binding”, similar to software development.

This is not always possible, since there are situations, where the combination of elements from different semantic definitions (different namespaces) depends on applications or context. Such a “late binding” can be done by inserting specific placeholders in a schema. The schema construct xsd:anyType is a typical example of a placeholder. In the context of the model, it can be used to specify extensibility of objects, features, or parameters. Using the very same method, schemas based on the concepts from the common model may be integrated into existing definitions. There is no restriction concerning the organizational layer of such an existing definition within the automation hierarchy. For example, device models according to (ISO 2003) may be source or target of extension, or definitions from the enterprise level like (ISA 2001) or (ISA 1995).

The methods described above allow extensibility at a schema level. However, extensibility can also be supported at an instance level. This means, that definitions are made in XML files rather than in schemas. A typical example is a mapping of codes to (perhaps language-depending) textual information. When such a coding is modelled as an object according to the Primitives, the instance file finally contains a number of instances of that particular object type. By using reference mechanisms as described above, these instances can be referenced to. Using the XPath addressing schema, fixed or open, strong typed or loosely defined, 1:1 or 1:n references can be modelled. This allows the required flexibility of descriptions. Furthermore, typed references can be used. They can be evaluated by a tool or even a style sheet, which are able to consider context information for interpreting (selecting and resolving) references. When XLink technology (W3C Consortium 2000b) is widely supported at a generic level, the model may be extended to allow XLink constructs in parallel to XPath or direct references.

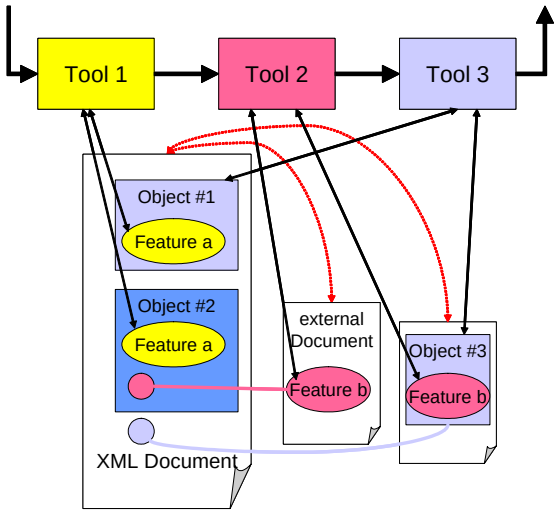

Fig. VI. A tool chain using distributed content.

Together with the application-specific identification of an object, complex tool chains may be realised, operating on the very same, perhaps distributed, description content. Each of the tools may access selected parts of the content, either complete objects or just features, may enrich it with specific information, and provide it for other tools of the chain. A

principle of such a tool chain is shown in Fig. VI. The possibly distributed character of the description content can be recognized, the common model's built-in mechanisms are used to provide consistency.

6. BUILDING BLOCKS

The extensibility concepts are widely used to create building blocks of re-usable information. Although a specification of information fragments and snippets is also possible with the existing XML technologies, its re-use is still problematic. This is especially caused by different modelling concepts used, which prevent a unique way of identification and reference. Furthermore, the typically closed description models lack the flexibility required. This has completely changed with the common model described above.

Consequently, all XML-based definitions within PROFIBUS and PROFINET environment will be set up using the common model. Using the extensibility concepts, integration into existing solutions can be done. Thus, a migration path from existing specific, closed solutions to those following the approach described above can be opened.

The goal is to provide a common set of such building blocks useful throughout the whole PROFIBUS and PROFINET environment, but not limited to. The set will consist of a combination of schemas and of instance files.

7. CONCLUSIONS

The use of XML descriptions in the automation domain has become state of the art. Because of the variety of applications, there's no best practice pattern on how to model content with XML. Clearly based on the use cases, different concepts have evolved, each of them designed with a specific context and viewpoint in mind.

As a result, there's no common relation between the different models, nor is a way for creating automatic tools for translation or data exchange. Thus, double definitions, double efforts in design, complex translation methods etc. reduce the benefits expected when introducing XML into the automation domain.

Taking the typical use cases for the automation domain into account, a common model for handling XML was developed for applications in PROFIBUS and PROFINET systems, but not limited to. As described above, it provides basic definitions and rules for modelling content. Its outstanding flexibility and extensibility allows an easy integration into existing approaches, and an easy creation of re-usable definitions as building blocks. Since there is no generic model for handling XML in the automation and control domain, it could be useful to discuss these concepts in other activities applying XML.

Besides its generic character, the model is open for further developments. When new technologies come into a mature and accepted stage, they will be integrated. Besides the already mentioned XLink, current developments from the Semantic Web area are promising candidates.

Finally, the model is currently being introduced into specification groups and standardization bodies in order to achieve a broad acceptance.

REFERENCES

FDCML (2002) *Field Device Configuration Markup Language*. FDCML 2.0 Specification, Version 1.0, http://www.fdcml.org.

IEC (2002) *IEC 61158 – Digital data communication for measurement and control – Fieldbus for use in industrial control systems.*

ISA (1995) *ANSI/ISA-88.01-1995 - Batch Control Part 1: Models and Terminology.*

ISA (2001) *ANSI/ISA-95.00.02-2001 - Enterprise-Control System Integration Part 2: Object Model Attributes.*

ISO (2003) *ISO 15745-3 Industrial automation systems and integration -- Open systems application integration framework -- Part 3: Reference description for IEC 61158-based control systems*.

ISO (2004) *ISO 10303 Industrial automation systems and integration - Product data representation and exchange.*

PI (2003). *PROFInet - Architecture Description and Specification.* PROFIBUS International, No. 2.202.

PI (2004). *XML@PROFIBUS*. PROFIBUS International, Guideline, No. 2.342.

W3C Consortium (1999). *XML Path Language (XPath), Version 1.0,* 16-Nov-1999, http://www.w3.org/TR/xpath/

W3C Consortium (2000a). *Extensible Markup Language (XML) 1.0* 06-Oct-2000, http://www.w3.org/TR/2000/REC-xml-20001006

W3C Consortium (2000b). *XML Linking Language (XLink) Version 1.0* 20-Dec-2000. http://www.w3.org/TR/2000/PR-xlink-20001220/

W3C Consortium (2001a). *XML Schema Part 1: Structures, W3C Recommendation,* 2-May-2001, http://www.w3.org/TR/xmlschema-1/

W3C Consortium (2001b). *XML-Signature Syntax and Processing*, W3C Proposed Recommendation, 20-Aug-2001, http://www.w3.org/TR/2001/PR-xmldsig-core-20010820/

Wollschlaeger, M., F. Geyer, D. Krumsiek, and R. Wilzeck (2003). XML-based Description Model of a Web Portal for Maintenance of Machines and Systems. *9th IEEE International Conference on Emerging Technologies and Factory Automation ETFA2003*, **Vol.1**, pp. 333-340.

Wollschlaeger, M., M. Thron, and R. Simon (2004). XML in Control Systems. *IEE Open Control Systems – The Importance of Industrial Standards*, Birmingham (UK).