Definition and Generation of Data Exchange Formats in AUTOSAR

Mike Pagel[1] and Mark Brörkens[2]

[1] BMW AG, Knorrstr. 147, 80788 München, Germany
mike.pagel@bmw.de
[2] Carmeq GmbH, Carnotstr. 4, 10587 Berlin, Germany
mark.broerkens@carmeq.com

Abstract. In this paper we present a methodology supporting the definition of data models on basis of a limited set of well-known UML features, thereby allowing these models to be created and discussed by a large group of domain experts. A transformation is then defined from such a platform independent UML model to XML schema, which exceeds the configuration possibilities of comparable approaches like XMI. This enables the generic reproduction of a wide range of existing XML languages and hence supports reverse-engineering legacy schemas and DTDs into well-structured UML models. The overview of an actual implementation of the generic methodology finally demonstrates the practical applicability or our approach. The work described in this paper is part of the AUTOSAR development partnership, an international effort to standardize automotive software infrastructure. The resulting XML schema is used today as the official AUTOSAR XML data exchange format.

1 Introduction

AUTOSAR (short for: automotive open system architecture) is an international development partnership [1] consisting of a multitude of car manufacturers, suppliers and tool vendors, defining concepts and workflows, how electronic automotive systems can be formally specified and processed. Currently, AUTOSAR is mainly focusing on software and addresses issues like hardware independence, design-by-contract, system scalability, reuse and so forth.

The definition of AUTOSAR concepts (which themselves are not in scope of this paper) in form of a metamodel leads to a domain specific language (DSL). While this language is specifically designed to describe distributed real-time software, it still is platform-independent in terms of processing platforms like XML, databases or a programming language. System descriptions written in this language must be interchangeable between various authoring, visualization and processing tools as well as the different organizational entities involved in an AUTOSAR-oriented project. One of the main goals of AUTOSAR is therefore the definition of an (automatically generated) XML-based data exchange format.

While generating XML schema from UML is not new, a number of problems were encountered with approaches and methods typically applied in the industry so far. For instance:

A. Rensink and J. Warmer (Eds.): ECMDA-FA 2006, LNCS 4066, pp. 52–65, 2006.

- XMI [2][3], OMG's specification how to map UML models to XML schema, lacks certain configuration possibilities, thereby preventing the reproduction of already existing XML schemas from reverse engineered UML models. Furthermore, XMI uses particular schema features like xsi:type, which in the XML community are discussed as problematic [4][5].
- The MSR partnership [6] defined an XML DTD to describe automotive systems. The underlying data structures have not been formally modeled; but instead were designed directly at DTD level.
- The ASAM association introduced another modeling approach for the ASAM ODX standard [7], leading to an XML description for automotive diagnostic and programming systems. The corresponding schema is in fact generated from a UML model. However, the applied UML profile is highly specific for the XML domain and therefore alleviates the applicability of the metamodel as an MDA PIM, e.g. to produce a database schema.

Our transformation of the AUTOSAR metamodel to XML schema exceeds the configuration capabilities of current approaches. It is defined as a set of transformation patterns and model markings. The actual implementation of our tool-chain is based on the Eclipse Modeling Framework (EMF).

Outline of this Paper. The next section gives an overview of how our approach is aligned with general MDA concepts. The remaining sections then follow the logical order of applying our methodology. We begin with the description of our concepts to define a platform independent model. The following section explains our requirements for a new transformation from UML to XML schema and specifies the configuration capabilities of our approach. Next, we describe our implementation of the schema generator. The paper closes with a final summary of our results and an outlook of possible next steps.

2 Alignment with MDA Concepts

Fig. 1 gives an overview how our modeling and generation approach is aligned with the concepts defined by MDA, which are shown as stereotypes of our corresponding AUTOSAR artifacts.

The AUTOSAR metamodel is a PIM. Its definition is independent of the tooling and processing platforms (e.g. programming languages, databases, XML formats, ...) that eventually will be involved in creating, editing or persisting AUTOSAR models. This platform independence is strongly required to allow automotive domain experts to contribute efficiently without consideration of how modeled concepts eventually will be mapped to a certain platform.

Our primary platform is XML schema, i.e. that we are generating the AUTOSAR XML schema from the AUTOSAR metamodel.

The mapping is defined as a set of transformations, which are combinations of type and instance mappings. For each relevant PIM type a template is defining the default purely type-based mapping. If a certain element in the AUTOSAR metamodel requires different transformation, this is controlled by assigning values to platform specific marks which leads to the annotated metamodel, a marked PIM. A modeling

guideline prescribes tagged values as the only UML feature allowed to express such PIM markings.

The target of our mapping is not a separate PSM. Instead, we follow the alternative path also mentioned in [8] by directly generating the actual XML schema file from the marked PIM, for two reasons: While research projects [9][10] start providing tools to perform model transformations, frequently as an implementation of the MOF QVT specification [11], we did not find sufficient transformation support as part of commercial UML tools available to AUTOSAR members. Having an explicit PSM does not seem to add much value to our methodology. Instead, performing the extra transformation step adds a potential error source, resulting from defects in the used tooling as well as problems introduced by improper usage.

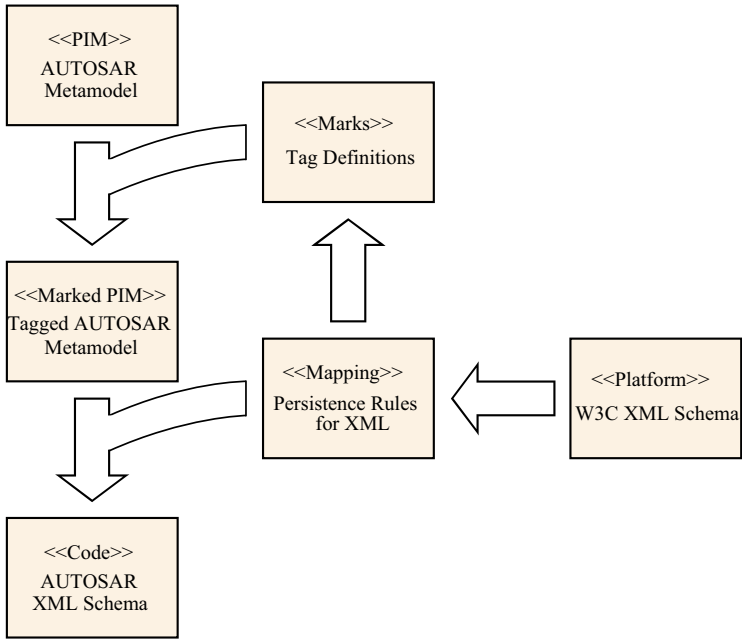

Fig. 1. The alignment between the MDA methodology and the AUTOSAR modeling and schema generation approach is shown in this figure. For each depicted AUTOSAR artifact the stereotype-like annotations indicate the corresponding MDA concept.

3 Definition of the AUTOSAR PIM

The language to describe AUTOSAR systems is a DSL enabling the definition of automotive software and hardware systems. It therefore contains specialized entities like certain types of software components that are responsible for handling hardware sensors or entities like communication protocols used on automotive bus systems. While the usage of UML to describe this metamodel was not disputed, a number of additional requirements had to be taken into account.

In the automotive industry the practical application of UML is not yet as widely adopted as in other, more traditional IT domains. To overcome this psychological barrier the AUTOSAR methodology had to limit itself in terms of used UML features. On the other hand, certain features required to describe AUTOSAR systems are not well supported in UML. In those cases, we had to extend the existing modeling capabilities. Finally, practical reasons like existing tool support forced us to sometimes deviate from pure MDA and UML approaches.

This section explains those limitations and extensions of UML.

3.1 Usage of a UML Subset

Even when using UML, still a number of alternative approaches can be considered for the definition of the AUTOSAR metamodel. First, the metalevel the AUTOSAR DSL is associated with or is build upon needs to be determined. Corresponding to OMG's four-layer metamodel hierarchy [12], AUTOSAR user models, e.g. the model of a concrete windshield wiper system, live on M1. Therefore, the underlying AUTOSAR metamodel that defines the language to express such M1 models is defined on M2.

To stay completely aligned with OMG's metalevel hierarchy, this suggests two well-known alternatives for the definition of the AUTOSAR metamodel: it can be defined as an instance of MOF [8], or as a UML profile. Unfortunately, both approaches have practical disadvantages.

The formal usage of MOF as modeling language was not well supported by tools when work on the AUTOSAR metamodel was started in the end of 2003. This means that limiting our modeling capabilities to the EMOF subset [8] was not possible through available tools. This is different today, as tools like IBM's RSM [13] and Sparx' Enterprise Architect [14] start providing metamodeling capabilities.

Creating a UML profile has the strong advantage that even standard UML tools are able to handle the corresponding UML instance models. However, tool support for creation and especially formal application of a UML profile was again too weak for us. While this also has been overcome by recent tools, the formal definition of such a profile is not trivial and naturally requires deep insight into UML's own metamodel. Such deep analysis of UML is typically not part of the daily business of automotive engineers and hence only sparsely available. Therefore, requiring the definition of a formal UML profile for the AUTOSAR metamodel was not possible.

As a consequence AUTOSAR is following a mixed approach. The original AUTOSAR metamodel is created as an instance of UML. From UML we only allow using basic class diagrams, essentially the concepts defined in EMOF or the UML infrastructure's Core::Basic package [12]. We further restrict the allowed set of modeling features, e.g. classes defined in the AUTOSAR model must not own any operations. These limited class-modeling capabilities are formally specified in form of its own UML profile. Since the AUTOSAR metamodel exists at M2, the underlying profile is defined at M3. Currently, the AUTOSAR metamodel itself is mapped to a formal UML profile by a small group of experts, who are involved in both, AUTOSAR development and UML inner workings. For simplicity and due to growing tool support the rest of the paper will assume the AUTOSAR metamodel residing at OMG's metalevel M2, i.e. either as direct MOF instance or in form of a UML profile, as shown in Fig. 2.

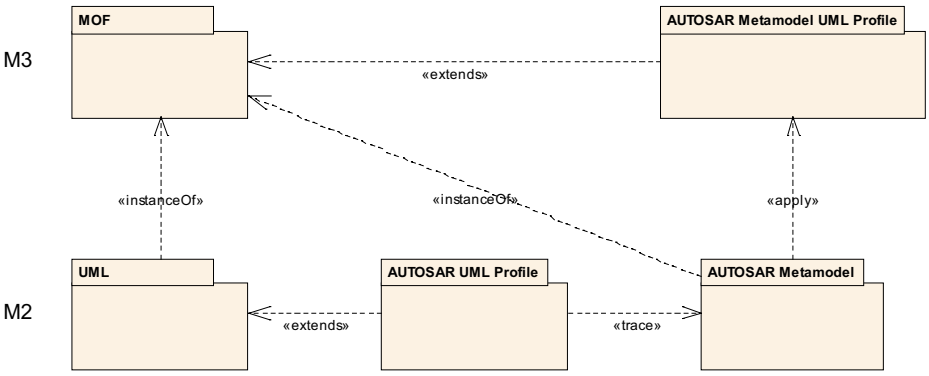

Fig. 2. The (AUTOSAR Metamodel) is effectively an instance of (MOF), which applies the (Metamodel UML Profile) to provide required extensions and enforce our limitations. Eventually, the (AUTOSAR Metamodel) is translated into its own (AUTOSAR UML Profile) to allow AUTOSAR modeling in standard UML tools.

3.2 Definition and Structure of PIM Markings

Platform-dependent information may be added to the model in form of tagged values only. Vice versa, tagged values are exclusively used to specify platform- dependent information. The names of the tag definitions are following a namespace scheme. E.g. a tag called xml.name specifies how a (platform-independent) model element will be called in a (platform-dependent) XML file, whereas a tag db.name could specify the same for a database platform. For an overview of available marks see table 1.

3.3 Stereotypes of the AUTOSAR Metamodel UML Profile

The AUTOSAR metamodel UML profile defines a handful of new stereotypes to either explicitly underline concepts already available in UML or to add modeling capabilities that are not part of standard UML. Those stereotypes are usually prefixed by ‘atp’ (AUTOSAR template profile; a synonym for AUTOSAR metamodel UML profile).

Types and Typed Elements. UML supports types and typed elements as one of the most fundamental modeling concepts expressed in the Types diagram of the UML infrastructure’s Core::Basic package [12]. While modeling of reusable types and their later use in form of roles and instances is typical in many IT domains, automotive software models are very often rather instance oriented. In those models, reuse is provided in form of creating copies and changing them if required.

One of the main goals of AUTOSAR in fact is the reuse of software and corresponding models. To make the distinction between types and typed elements more explicit, we enable this feature at M3, in the metamodel profile, in form of three new stereotypes:

– «atpType» is applied to classes in the AUTOSAR metamodel that are defining a type. This directly corresponds to classes being derived from UML’s Type metaclass, in most cases those classes are refined forms of UML’s StructuredClassifier.

– «atpPrototype» is applied to all typed classes. Again, this is a very explicit form of deriving a class from UML's TypedElement metaclass, in most cases these classes correspond to UML's Property. The name prototype was chosen because of compatibility concerns regarding MSRSW [6], an existing description format for automotive electronic systems. Additionally, since classes with stereotype «atpPrototype» are roles of a certain type, they in fact *are* prototypes for instances that are created at runtime.
– «isOfType» is finally applied to the relation between a prototype and a type. Every «atpPrototype» can reference at most one «atpType», just as in UML.

Through those stereotypes modelers are forced to explicitly distinguish types and typed elements. The following figure shows a small excerpt from the AUTOSAR metamodel where the aforementioned stereotypes are applied.

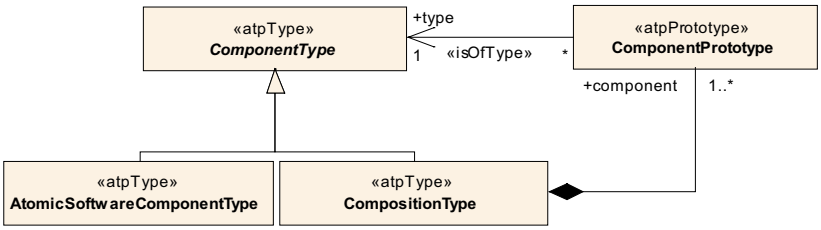

Fig. 3. The (AUTOSAR Metamodel Profile) defines new stereotypes for the explicit definition of types and the corresponding typed elements (called prototypes here for historical reasons). The figure shows a small example from the AUTOSAR metamodel demonstrating the stereotype application.

Deep References to Parts of Parts. The fact that AUTOSAR strongly supports the definition/usage dichotomy through its types and prototypes introduces a challenge in situations, where an element needs to be referenced that is deeply nested in a part-of-parts hierarchy as shown in Fig. 4. To ease understanding the chosen example is not a software problem but describes a bicycle; not just any bicycle, but yours! Your bike consists of two wheels of a certain type. Those wheel types in turn consist of their own parts like a tube and a tire of some kind. If you now take your bicycle out on a trip and blow the front tube you later will need to tell a mechanic what to fix. You can't just ask to repair a tube of type TubeX, because possibly every RacingWheel in the shop, and for sure the two wheels on your bike both have that tube. If you mention that the tube of a frontWheel needs to be fixed, there may still be multiple RacingBikes having this part. The work order is precise enough once you specify that the tube of the frontWheel of yourBike needs to be repaired as indicated in Fig. 5. In technical terms TubeX, TireY, RacingWheel and RacingBike are all types, while tube, tire, front- and rearWheel as well as yourBike are usages of those types. A type consisting of parts of certain other types, which in turn consist of more parts and so on is a typical pattern in object-oriented languages. The bike example shows that for an exact reference of a leaf (or intermediate) part in the hierarchy the containing parts must also be specified. In AUTOSAR

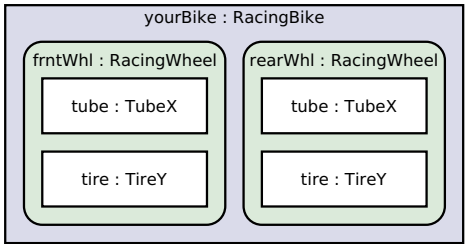

Fig. 4. Example for a typical part-of-parts hierarchy. A (RacingBike) is a type of bike that contains two wheels: (frontWheel) and (rearWheel). Those two wheels are usages of the same wheel type (RacingWheel). Some parts of that wheel type are a (tube) of tube type (TubeX) as well as a (tire) of tire type (TireY).

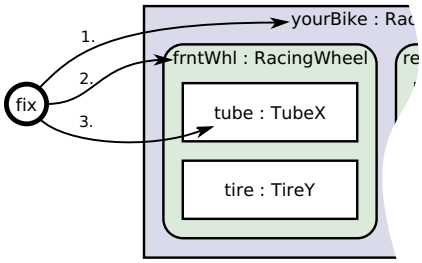

Fig. 5. To refer to a particular tube, the owning wheel as well as the bicycle must be specified. Complete references to parts-of-parts require the containing context. In AUTOSAR this is called in instance reference and consists of an ordered list of part references.

we find this requirement e.g. in case of a software component hierarchy. Composite components may consist of further composites and so on, which eventually consist of what in AUTOSAR is called an atomic software component. Only those leaf components contain actual code and use up resources like CPU time or memory, and exactly those leaf components must be deployed to the processing nodes (ECUs, electronic control units) of the system, i.e. they must be referable from a top level deployment element. UML 2 addresses this requirement only for exactly one hierarchy level in case of the UML connector connecting two ports of a structured classifier's parts: in addition to the port which is defined by its owning type the UML connector end also references the corresponding part through the attribute ConnectorEnd.partWithPort [15]. However, AUTOSAR requires this issue to be addressed for the general case of arbitrary depth. Therefore, we extended the UML 2 approach by not just specifying a single contextual part, but an ordered list of those, starting with the outermost and ending with the innermost part, just as shown in Fig. 5. A short form of this reference type is presented in Fig. 6 to once more allow our domain experts to express their concepts in a convenient and simple way. The corresponding stereotype «instanceRef» for the association got its name from the fact that the reference is indeed specifying an instance-like occurrence of the reference target instead of just a part in the context of its owning type. A

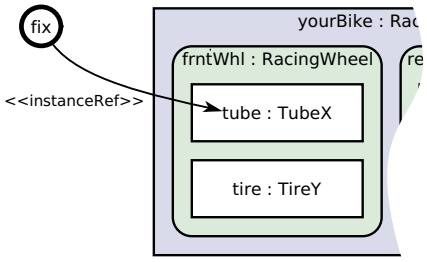

Fig. 6. AUTOSAR introduced an abbreviation for instance references in form of a regular association stereotyped («instanceRef»)

comparable motivation led to a similar concept in SysML in form of the SysML «Binding» stereotype, which also allows referencing a list of parts through the Binding.path attribute [16]. However, the SysML solution targets a different use-case.

4 Transformation of the PIM to XML Schema

This section describes our transformation of the AUTOSAR metamodel to an XML schema. Especially we motivate why and how our approach exceeds the possibilities provided by existing methods like XMI.

4.1 Design Patterns of the AUTOSAR XML Schema

Several approaches for translating UML or MOF based metamodels into XML Schema or XML DTD [3][2][17] have been evaluated. None of them fulfilled all requirements on the AUTOSAR data exchange format, the most important of which are listed below:

- unambiguous mapping between instances of the XML Schema and instances of the AUTOSAR metamodel,
- reuse of XML patterns well established in the automotive domain, and
- support of tools in the XML and the model domain.

The AUTOSAR XML schema fulfills these requirements by combining the strengths of several existing XML patterns: Our XML schema is a dialect of XMI 1.2 that is extended by legacy XML patterns, which are harmonized between several data exchange formats frequently used in the automotive industry.

The following sections describe the requirements and how they are fulfilled in more detail. After that the configuration possibilities and translation rules are explained.

Unambiguous Mapping. The seamless exchange of data between tools requires unambiguous mapping of information stored in the model domain to the XML domain and vice versa. AUTOSAR follows the concept used in XMI: By default all navigable

association ends, attributes and types in the model domain are explicitly represented by XML elements or attributes in the XML domain.

The default mapping rules used by AUTOSAR are inspired by XMI 1.2: navigable association ends, attributes and types in the metamodel are all represented as XML elements. Fig. 7 shows a typical metamodel fragment and an instance of the AUTOSAR XML schema created using the default mapping rules. The type information is required in order to manage inheritance: it must be possible to distinguish between ClassB and SubClassB. Additionally, the names of the navigable association ends (roleB1 and roleB2) are required in case of multiple associations with the same type. In case a model fragment doesn't make use of multiple association or inheritance it is allowed to overwrite the default mapping rules, as explained later in this section.

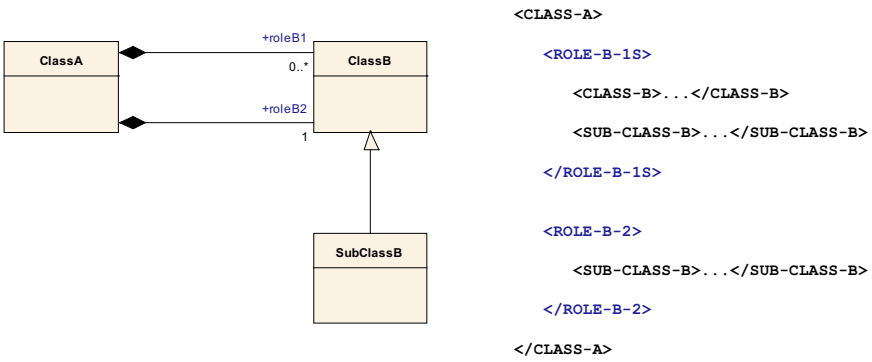

Fig. 7. A typical metamodel fragment, including classes, multiple composite associations, and a corresponding sample instance of the AUTOSAR XML schema created using the default mapping rules

Support for Tools in the XML and Model Domain. The AUTOSAR data exchange format will be used by a wide variety of different tools such as primitive XML editors, legacy tools and graphical tools explicitly optimized for the AUTOSAR methodology. Most tools internally implement their own data model with a structure different from the AUTOSAR metamodel. While XMI strives to satisfy the requirement to find a very simple transformation from a metamodel to its XML representation, we see no great advantage in pursuing this goal, as the resulting direct transformation is not possible for tools and their proprietary models. We therefore trade the straightforwardness of the XMI transformation against high configurability.

Additionally, we limit used XML schema features to a subset that is generally accepted within the XML community and is implemented by most off-the-shelf XML tools and libraries. E.g. we do not use XML schema inheritance because it is limited to single inheritance and makes use of features like xsi:type, the usefulness of which is debated among XML experts [4][5].

Inheritance in the AUTOSAR metamodel is mapped to the AUTOSAR XML schema using the copy-down-approach of XMI 1.2. However, instead of repeating inherited

properties in the declaration of XML elements, (see e.g. the repetition of ModelElement.name in the XMI 1.2 MOF DTD [2]) we use element and attribute groups, which are referenced if needed. Polymorphism is made explicit in the schema by listing all concrete subtypes.

Support of Existing DTD Based XML Formats. Some concepts used in AUTOSAR are already well defined by existing XML formats used in the automotive industry. For selected contents the translation of the AUTOSAR metamodel to the XML Schema shall be flexible enough to reproduce such a standardized format. AUTOSAR addresses this requirement by providing the advanced configuration possibilities explained below.

Configuration of XML Schema Production Rules. The AUTOSAR XML schema production rules are configured by UML tagged values attached to the metaclasses and owned properties of the AUTOSAR metamodel. Default tagged values are implied in (the very typical) case, where no explicit values have been added to a model element. Tagged values on roles (references, composition and attributes) control which XML elements or attributes are generated for representing the given role. Additionally order and multiplicity is configurable.

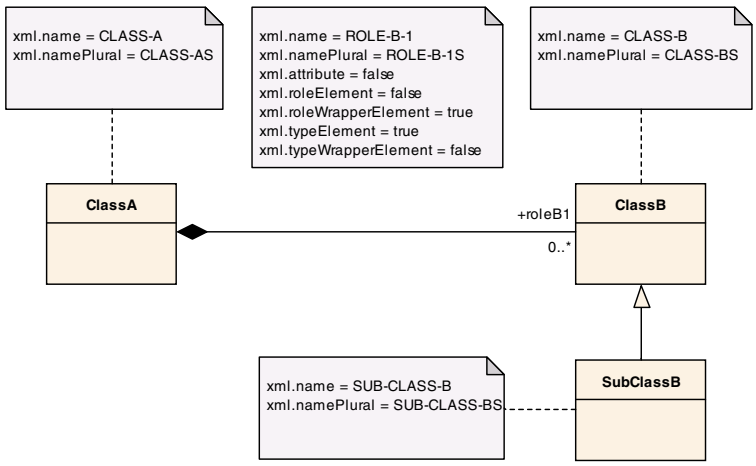

Fig. 8. The platform independent UML model is annotated with a set of predefined tagged values: our model marks. The tags specify how a model entity is translated to the XML schema platform, e.g. whether an XML element or XML attribute has to be generated, in which order XML elements have to appear, or whether wrapper elements for multiple classes need to be generated.

Fig. 8 shows the default configurations for a composition relationship with unbounded upper multiplicity. Table 1 lists the most important tagged values that are part of our schema production configuration.

Table 1. List of the most relevant tagged values used in our transformation from the AUTOSAR metamodel to XML schema. The values refer to roles as synonym for UML properties.

Tag name	Applicable to	Description
xml.name	role, class	Provides the name of a schema fragment representing the role or class.
xml.namePlural	role, class	Provides the plural name of a schema fragment.
xml.attribute	role	If set to true, the role is represented as attribute. This tag is only applicable for roles typed by a primitive datatype with an upper multiplicity of 1.
xml.roleElement	role	If set to true, the xml.name of the role shows up as XML element.
xml.roleWrapperElement	role	If set to true, the xml.namePlural of the role shows up as XML element. This XML element is typically generated in case the multiplicity of the role is greater than 1.
xml.typeElement	role	If set to true, the xml.name if the role's type shows up as XML element.
xml.typeWrapperElement	role	If set to true, the xml.namePlural of the role's type shows up as XML element.

The following listing shows how the aforementioned metamodel fragment is mapped to XML schema:

```
<xsd:element name="CLASS-A" type="CLASS-A"/>

<xsd:complexType name="CLASS-A" abstract="false" mixed="false">
   <xsd:sequence>
     <xsd:group ref="CLASS-A"/>
   </xsd:sequence>
 </xsd:complexType>

<xsd:group name="CLASS-A">
   <xsd:sequence>
     <xsd:element name="ROLE-B-1S" minOccurs="0">
       <xsd:complexType>
         <xsd:choice minOccurs="0" maxOccurs="unbounded">
           <xsd:element name="CLASS-B" type="CLASS-B"/>
           <xsd:element name="SUB-CLASS-B" type="SUB-CLASS-B"/>
         </xsd:choice>
       </xsd:complexType>
     </xsd:element>
   </xsd:sequence>
 </xsd:group>

<xsd:complexType name="CLASS-B" abstract="false" mixed="false">
   <xsd:sequence>
     <xsd:group ref="CLASS-B"/>
   </xsd:sequence>
 </xsd:complexType>

<xsd:complexType name="SUB-CLASS-B" abstract="false" mixed="false">
   <xsd:sequence>
     <xsd:group ref="CLASS-B"/>
     <xsd:group ref="SUB-CLASS-B"/>
```

```
  </xsd:sequence>
</xsd:complexType>

<xsd:group name="CLASS-B">
  <xsd:sequence>
    ...
  </xsd:sequence>
</xsd:group>

 <xsd:group name="SUB-CLASS-B">
  <xsd:sequence>
    ...
  </xsd:sequence>
 </xsd:group>
```

5 Implementation of the Schema Generator

Our tool chain links a number of publicly available modeling and processing tools. The complete process is depicted in Fig. 9.

We have two important sources defining AUTOSAR concepts: (a) the AUTOSAR workgroups directly specifying particular content of the metamodel and (b) an existing standard for automotive software systems, the ASAM MSRSW harmonized objects [7]. The original AUTOSAR concepts were directly modeled utilizing our UML subset with Sparx System's Enterprise Architect. In order to incorporate the MSRSW models, which at that time did not exist in UML, but were available as DTD only, we realized a simple reverse-engineering script, which creates a model compliant with the AUTOSAR modeling guidelines from the MSRSW DTD. Explicit import statements starting at the original AUTOSAR metamodel specify, which parts of the complete MSRSW need to be available as part of the joint AUTOSAR metamodel.

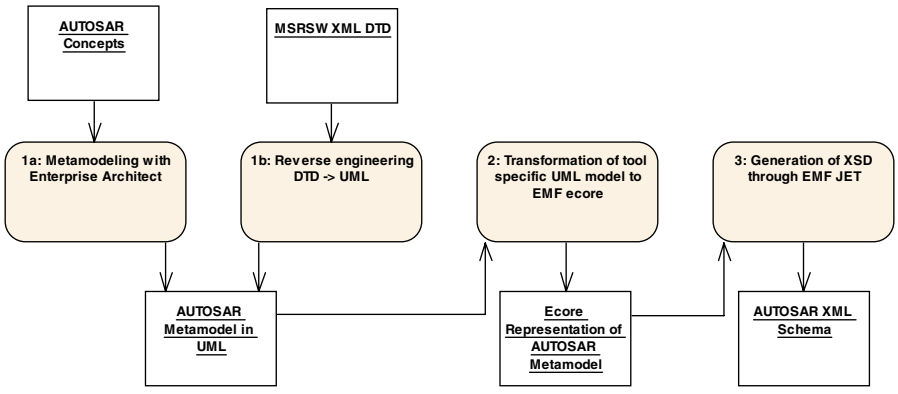

Fig. 9. Overview of the implemented AUTOSAR metamodel tool-chain. The applied process steps are numbered in their order of execution. (1a) and (1b) indicate the two possible sources for models: original AUTOSAR concepts as well as existing standard formats like the MSRSW DTD.

The second processing step transforms the modeling tool dependent metamodel into an independent Java representation. Here we found the Eclipse Modeling Framework (EMF) [18] most useful, with a very clean API and extremely good support for metamodeling tasks through arbitrarily deep access to all metalevels of a model in a single application. The conversion step includes a number of mechanisms that allow for graceful degradation in case the model is not fully following the AUTOSAR metamodel rules. While the final model of course must be compliant, the sheer number of people contributing to it requires a resilient algorithm in order to create and verify results early in the process.

Finally, we use EMF's Java Emitter Templates (JET) to translate the EMF based metamodel into its final representation: the AUTOSAR XML schema.

6 Summary and Outlook

The Model Driven Architecture approach is well suited for defining the AUTOSAR domain specific language and the corresponding data exchange format. While using standard UML modeling techniques, we limit the expressive power of UML to a degree allowing automotive engineers to actively contribute to the metamodel. This is further supported by the strict platform independence of the model. No XML knowledge is required to work on the AUTOSAR metamodel and finally get new concepts into the exchange format.

The metamodel is optionally marked up with tagged values for configuring the translation to XML schema. The configuration possibilities allow for powerful adaptation of the resulting XML format. If the default XMI 1.2-like translation is not sufficient it may be customized to suite individual needs. This allows for reproduction of XML fragments out of the AUTOSAR metamodel, which are already established in the automotive domain as part of other standard data exchange formats. We proved the applicability of our approach by describing the Java implementation of our EMF-based schema generator. Future work may happen in the following areas:

- Additional transformations can be added. For instance, we are currently creating generators that realize persistence code to read and write the format defined by AUTOSAR. Since commercial tools typically will not be based directly on the AUTOSAR metamodel, but much rather will instantiate their own, tool- optimal model, such generators will therefore be specific for the target tool environment.
- Since many tools and projects start supporting formal model transformations as prescribed by the OMG QVT specification we will start evaluating available platforms and eventually switch over to begin using more formal methods instead of our handcrafted transformation code. This applies to both, the description of the transformations as well as the actual implementation.
- The AUTOSAR metamodel is annotated with OCL constraints [19] where required. But as of now, we are not automatically evaluating those. With the recent releases of more capable OCL processors we will start to implement automatic constraint checking in upcoming versions of our tool-chain.

References

1. AUTOSAR: Homepage http://www.autosar.org (2006)
2. OMG: XML Metadata Interchange (XMI) specification version 1.2 (2002)
3. OMG: XML Metadata Interchange (XMI) specification version 2.1 (2005)
4. Dubinko, M.: chapter 4. In: XForms Essential, XML Schema in XForms. Volume 1. O'Reilly Media Inc. (2003)
5. Walsh, N.: xsi:type train wreck. In: Norman Walsh Weblog. Volume 7. (2004) http://norman.walsh.name/2004/01/29/trainwreck.
6. Manufacturer Supplier Relationship (MSR): MSRSW V2.2.2, element and attribute documentation (2002)
7. Association for Standardization of Automation- and Measuring Systems (ASAM): ASAM MCD-2D (ODX) version 2.0, data model specification (2004)
8. OMG: Meta Object Facility (MOF) specification version 2.0 (2006)
9. IBM alphaWorks: Model transformation framework (2006) http://www.alphaworks.ibm.com/tech/mtf.
10. UMT-QVT: an open-source project targeting the QVT RFP (2006) http://umt-qvt.sourceforge.net/.
11. OMG: Meta Object Facility (MOF) query/view/transformation specification version 2.0 (2005)
12. OMG: UML Infrastructure specification version 2.0 (2004)
13. IBM Rational: Rational software modeler product page (2006) http://www-306.ibm.com/software/awdtools/modeler/swmodeler/.
14. Sparx Systems: Enterprise architect product page (2006) http://www.sparxsystems.com.au/.
15. OMG: UML Superstructure specification version 2.0 (2005)
16. SysML Partners: Systems modeling language specification version 1.0 alpha (2005)
17. Carlson, D.: Modeling XML Applications with UML, Practical e-Business Applications. Addison Wesley (2001)
18. Budinsky, F., Steinberg, D., Merks, E., Ellersick, R., Grose, T.: Eclipse Modeling Mramework, A Developer's Guide. Addison Wesley (2005)
19. OMG: UML OCL specification version 2.0 (2005)