XML-BASED MODELING LANGUAGES AND DATA BINDING
FOR COLLABORATIVE DESIGN IN MULTIDISCIPLINARY TEAMS

Martin Hoppen  
Malte Rast  
Michael Schluse  
Juergen Rossmann  
Institute for Man-Machine Interaction  
RWTH Aachen University  
Ahornstrasse 55, 52074 Aachen, Germany  
E-mail: hoppen@mmi.rwth-aachen.de

Lars Pfotzer  
FZI Research Center for  
Information Technology  
Haid-und-Neu-Strasse 10-14  
76131 Karlsruhe, Germany  
E-mail: pfotzer@fzi.de

Thomas Meschede  
TU Berlin, Department of  
Aeronautics and Astronautics  
Marchstrasse 12-14  
10587 Berlin, Germany  
E-mail: thomas.meschede@tu-berlin.de

KEYWORDS
Collaborative Environments, Modeling Languages, 3D Simulation, XML, XSD, Data Binding

ABSTRACT

3D simulation applications are often developed or employed in distributed and multidisciplinary teams. An important basis is a common language for data modeling and exchange to simplify or even permit a common understanding. The Extensible Markup Language (XML) provides an excellent basis technology to define a common modeling language. Such a language can either be customized to a specific problem domain like modular and reconfigurable satellites and their on-orbit-servicing (OOS) in a distributed R&D space project. Alternatively, a standardized modeling language like X3D or AutomationML can be used if it fits an application’s requirements. A common modeling language shall, however, not only be used for data exchange, but also as an application schema for participating systems to avoid ”friction losses” due to translations. For that purpose, XML data binding techniques can be used to integrate XML schemata and data into object-oriented runtime systems. This allows for a tighter integration than generic interfaces like the Document Object Model (DOM) or the Simple API for XML (SAX).

INTRODUCTION

When collaboratively working on 3D simulation projects in distributed teams, special care has to be taken regarding the incorporated data. A well-defined, common data modeling language is needed to allow for consistent data management and exchange. It must be flexible and particularly extensible to accommodate to the different requirements of various applications and heterogeneous teams using different tools. For that purpose, XML provides an excellent basis technology. On the other hand, for applications from the 3D context like Geographic Information Systems (GIS), Computer-Aided Design (CAD), or other 3D software, an object-oriented modeling is advisable, as the corresponding data usually consists of a huge number of hierarchically structured parts with interdependencies (Elmasri and Navathe 2010).

To integrate both requirements, XML data has to be made available within object-oriented runtime systems. This can be done generically using techniques like DOM or the SAX. More intuitive is the so-called XML data binding approach. Here, the schema of the XML data is mapped to object-oriented classes and their attributes and XML data is made available in terms of corresponding objects. However, the challenge is the lack of standard mappings due to ambiguities in between the two domains (XML and object-orientation). Thus, the intended data model can only be preserved when the modeling conventions within the XML schema are understood and reasonably translated to object-oriented structures.

In this paper, we present two use cases of XML-based modeling languages and corresponding data binding solutions from the context of 3D simulation projects. The first considers the case of a customized base language for modeling general object-oriented 3D data in XML. Here, modeling conventions and object-oriented mappings are adopted from the well-defined Geography Modeling Language (GML). As a second use case, we present XML data binding approaches for established standard modeling languages using the example of the VRML successor X3D and AutomationML, a language for modeling plant engineering data.

As an example for the customized modeling language, we present a multidisciplinary R&D space project. Today’s satellites are mostly monolithic systems without the possibility for maintenance, repair or reconfiguration. The lifetime of individual components or the mission duration determines the lifetime of the whole satellite. At the end of its lifetime, the satellite becomes space debris that endangers other satellites, the international space station or even populated areas on the earth. To overcome this problem, the project iBOSS
(Weise et al. 2012) develops a concept for modular satellites that can be reconfigured, tugged to other orbits or deorbited in a controlled crash. Figure 1 shows the idea of the reconfiguration of such a modular satellite under investigation in a virtual environment.

![Satellite Model based on the iBOSS Concept (Weise et al. 2012) in Virtual Reality](image)

Part of the project is the generation of an extensible catalog describing all satellite components and payloads. Satellite configurations can be constructed from this catalog and analyzed with respect to technical constraints or reconfiguration scenarios. With a common XML-based modeling language for this catalog and the models constructed from it, the iBOSS project partners can effectively join their expertise on spacecrafts, robotics, lightweight construction, 3D simulation, and Virtual Reality.

**STATE OF THE ART**

XML is a widespread format for data modeling and exchange. XML data builds a tree-like hierarchical structure with complex elements as inner nodes and simple elements as leaf nodes.

Given a schema description, an XML document is called valid if it is well-formed (regarding the XML standard) and its contents comply with the schema. Common methods for describing an XML schema are Document Type Definitions (DTDs) and XML Schema Definitions (XSDs) (Skulschus and Wiederstein 2004). For various reasons, including their non-XML syntax, DTDs have been superseded by the more powerful yet more complex XSDs. The XML Schema Definition Language for describing an XSD is based on XML itself providing the advantage of tool reuse and rich modeling features. An XSD can define global elements for documents, complex or simple types, complex or simple content models for types, local elements and attributes of complex types (including their type and multiplicity), inheritance of types (by extension or restriction), substitutability between elements, and abstractness of elements and types.

To generically access and process XML documents, two common methods exist. Using DOM, an XML document is completely parsed into a tree structure in memory where it can be accessed and manipulated using a standard API. While this approach is convenient, it may be bad for large documents. Another method is SAX, an event-based approach where the calling program is notified of each start or end tag providing stream-based processing.

Using DOM or SAX, XML data can be accessed in a very generic way. A more intuitive approach, however, is to map the XML-represented data and the corresponding schema to an object-oriented representation, i.e., to object and class structures of the utilized runtime system. Such a mapping is called XML data binding (Bourret 2001; 2011). In a process called unmarshalling or deserialization, an XML document's content is mapped to corresponding objects. To provide the necessary object-oriented structures, appropriate classes and attributes have to be derived, typically from a corresponding XSD. Different tools for XML data binding exist. However, the mapping from XSD to object-oriented structures is not as straightforward as it may seem.

This so-called impedance mismatch can be depicted with the example of XSD complex types and elements and their ambiguous mapping to the object-oriented class concept. Figure 2 gives a simplified XSD metamodel excerpt. On schema level, an Element has a ComplexType. The latter defines inheritance and a content model for the former. Both can independently be either abstract or not. On XML document level, instances of elements (ElementInstance) are used. Different approaches can be used to map Element and ComplexType to object-oriented class(es). A detailed survey can be found in (Bourret 2001).

![Simplified XSD Metamodel: Relationship between XSD Complex Type and XSD Element Definitions as well as Element Instances in Documents](image)
As a conclusion, a completely generic mapping between XML/XSD and object-oriented structures is inappropriate. Instead, in practical modeling languages, a limited set of modeling concepts and coding conventions is used for which specific semantics are defined. This can be seen in different standardized modeling languages in various fields of application like X3D for 3D and Virtual Reality data (Web3D Consortium 2014, Schwellenbach 2014), AutomationML for industrial automation (AutomationML association 2014, Konieczny 2014, Drath et al. 2008), and GML for geography information systems (Open Geospatial Consortium (OGC) 2014).

X3D is the XML-based successor of the Virtual Reality Modeling Language (VRML). An X3D document (or model) contains a scene graph describing the structure of the scene. X3D’s object model comprises complex objects called nodes and their properties called fields describing a node’s current state. A field either holds simple property values (that can also be used for event interchange) or contains other nodes. The latter is the basis for building up scene graph hierarchies. Using the special DEF and USE statements, nodes can be given unique names (DEF) by which they can be referenced (without copying) from other nodes (USE) allowing a node to have more than one parent. X3D object types are structured by inheritance in an interface hierarchy where only leaf classes are non-abstract. X3D’s XML encoding is defined by an XML Schema Definition.

The Automation Modeling Language (AutomationML) is an XML-based language for modeling and interchanging engineering data for automation technology. It combines the existing languages CAEX (Computer Aided Engineering Exchange) for object structure and language integration (Fedai and Drath 2005), COLLADA (Collaborative Design Activity) for geometry and kinematics, and PLCopen XML for logic. CAEX provides various concepts of object-oriented modeling.

GML provides basic structures for modeling and interchanging geographic data using XML. It usually serves as a foundation for customized modeling languages or application schemata, e.g., CityGML (CityGML 2014) for city data or ForestGML (Rossmann et al. 2013a) for forest data. For that purpose, it provides base types and well-defined modeling conventions with specific semantics.

USE CASE 1: CUSTOMIZED MODELING LANGUAGE

As we have seen, there is a variety of XML-based languages for specific purposes. In our iBOSS application example, we have to combine spacecraft design, robotics and 3D simulation. None of the existing XML-based languages exactly fits our needs. Therefore, we developed a custom modeling language for modular satellites and their robotic on-orbit reconfiguration. In order to reuse our concept for other application scenarios, we separated the modeling conventions in three levels in the following sections. The basis is a generic modeling language for representing object-oriented data in XML. On the second level, we add advanced techniques that may be of use in various engineering fields and, finally, we add an application specific layer.

**General Modeling Conventions**

As previously stated, in general, the concepts of XML and object-orientation do not match one-to-one. However, in this context the aim is not to map arbitrary XML documents with each and every possible XML modeling technique to an object-oriented representation and vice versa (also called round-trip). Instead, a common schema definition is agreed on, where modeling techniques can be limited and corresponding semantics be defined. For this purpose, the approaches from the GML specification as mentioned above can be adopted. In a generalized form, GML’s UML mappings are the conceptual basis for the following XSD modeling conventions.

One convention is to always define pairs of XSD global elements and corresponding complex types (Listing 1). In an object-oriented context, such a pair is mapped to a single class. Its base class is defined by the corresponding complex type’s extension base type. XML child elements are used to model the class’ attributes. In contrast, XML attributes are rather used for special properties of class attributes or objects like ids, id-based references or units of physical quantities. In the example, a class Satellite is defined using a global element definition and a corresponding complex type. The class is derived from class Spacecraft using the extension mechanism on type level. Here, a Satellite has two properties: an orbit (of type string) and a list of components (containing Component objects).

Listing 1: Class Definition in XSD

```xml
<xs:element name="Satellite" type="SatelliteType"/>
<xs:complexType name="SatelliteType">
    <xs:complexContent>
    <xs:extension base="SpacecraftType">
        <xs:sequence>
            <xs:element name="orbit" type="xs:string"/>
            <xs:element name="components" type="ComponentCompositionType" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

In a mechanism for typed reference properties, one can differentiate between references with or without transitive deletion characteristics. The former corresponds to UML composite aggregations while the latter to UML shared aggregations or "normal" UML associations. Other than UML, relations are typically modeled without independent association
types but rather using references from one object to another. Also adopted from GML, two modeling techniques for such references are used.

As a component is usually only referenced by one composite at a time, the proposed composite reference mechanism only allows inline definition of target elements (Listing 2).

Listing 2: Composite Reference Type in XSD

```
<x:schema name="ComponentCompositionType">
  <x:element minOccurs="0" maxOccurs="unbounded">
    <x:element name="Component" />
  </x:sequence>
</x:schemaType>
```

In contrast, for shared aggregations or simple cross references, targets may be referenced from several properties. Thus, an id-based mechanism is necessary. Adopted from GML, a generic reference type is defined that uses the XML Linking Language (XLink) (The World Wide Web Consortium (W3C) 2014) to refer to the target of the reference (Listing 3).

Listing 3: Class Definition with Non-Composite Reference in XSD

```
<x:schema name="Component" type="ComponentType"/>
<x:schemaType name="ComponentType">
  <x:element name="neighbor" type="ReferenceType" minOccurs="0">
    <x:annotation>
      <x:appinfo>
        <targetElement>Component</targetElement>
      </x:appinfo>
    </x:annotation>
  </x:element>
</x:sequence>
</x:schemaType>
```

An exemplary instance document conforming to the presented XSD definitions is given in Listing 4. The id XML attribute is defined in a super base class to consistently identify objects. The mass property shall be derived from a base class using an additional init XML attribute (see next section).

Listing 4: Example for Instance of Class Satellite

```
<Satellite id="satellite1">
  <mass unit="kg">1000</mass>
  <orbit>LEO</orbit>
  <components>
    <Component id="component1">
      <neighbor xlink:href="#component2" />
    </Component>
    <Component id="component2">
      <neighbor xlink:href="#component1" />
    </Component>
    <Component id="component3" />
  </components>
</Satellite>
```

Advanced Modeling Techniques

Besides the aforementioned basic modeling rules, some more advanced techniques were also integrated into the approach.

A variation of the previously introduced document-internal references are document-external references. In the context of 3D simulation models, such references can be used as an include mechanism for sub models from a catalog XML file. References are usually defined similar to compositions because sub models are interpreted as being instantiated as a part of a composite.

Another more advanced modeling technique similarly used in GML is to define physical quantities with units. Within the base schema, special derived simple types are defined, which combine a floating point value with a unit enumeration. An example is given in Listing 4 (mass).

Application-specific Modeling

To support integration, a common data modeling language was defined using an XSD following the aforementioned conventions. This schema is used to create valid XML documents for data exchange between project partners. To modularize construction, the iBOSS application schema defines an abstract satellite building block. It has common properties like position, orientation, and its interfaces. An interface is another type that can be used to logically connect blocks. Therefore, an interface element has a non-composite reference to its neighbor interface it is connected to. Using the building block base type, specialized blocks like a thruster block or a payload block are derived. A catalog XML file is used to offer a predefined choice of building block instances and components.

Currently, we use a component catalog that is referenced from the building block catalog, which again is referenced from actual satellite models (see figure 3).

![Figure 3: Structure of Schema Definitions and Component / Building Block Catalogs](image-url)
All catalog and model files comply with the application-specific XSD, based on the generic XSD, and can be nested in unrestricted depth. References to objects in submodels can be encoded as idpaths, e.g., `xlink:href="#id1#id2#id3"`, where `#id1` references a submodel in the current file, `#id2` references a subsubmodel in the submodel and `#id3` references the target object in the subsubmodel. Other file types can be included as well, e.g., for geometric models (X3D, STEP, etc.). It depends on the software tool and its import capabilities whether these includes are resolved.

### 3D Simulation and Virtual Reality System

In the context of the R&D project iBOSS, data is modeled according to the modeling conventions presented in the last sections. To demonstrate the usability, we present three tools for different purposes that access the data with XML data binding approaches.

In 3D Simulation, data is typically structured hierarchically, i.e., in a scene graph. To represent arbitrary spatial and logical structures, the Versatile Active Simulation Database (VSD) offers graph-based object-oriented data structures. The VSD is part of the Virtual Environments and Robotics Simulation System (VEROSIM) (Rossmann et al. 2013b) we use in our projects (Figure 1).

VSD provides a meta system to access its schema (i.e., a reflection API). This facility is needed for dynamic XML data binding as described in the next section. In Figure 4, its (simplified) structure is shown. Each object class in VSD is represented by a `MetaInstanceOf` containing `MetaMethods` and `MetaProperties` for describing its methods and attributes. Simple data types are represented by `MetaTypeVals`.

![Figure 4: Simplified Structure of VSD’s Meta System.](image)

XML Data Binding

To integrate XML schema and data into VEROSIM, an XML data binding approach was realized. The XML schema file is used to adopt the schema in VSD. The pairs of XML global elements and complex type definitions are either mapped to existing VSD `MetaInstances` or new ones are created (and mapped as well) where necessary. Accordingly, XML child element definitions are mapped to VSD `MetaProperties`. These can either be value or reference `MetaProperties`. For value `MetaProperties`, the XML type is translated to the corresponding VSD `MetaTypeVal`. E.g., an `xs:integer` type is mapped to an `int` within VSD. Likewise, typed references are identified in their XSD representation and mapped to `MetaProperties` with the appropriate `MetaInstance` target type and with or without transitive deletion characteristics. The special types for physical quantities with units are mapped to value `MetaProperties` with units.

After adopting the schema from an XSD file, corresponding XML files can be unmarshalled to VSD instances. For each XML element, a loading mechanism retrieves the corresponding `MetaInstanceOf` from the schema mapping. Accordingly, for child elements, corresponding `MetaProperties` are retrieved. The `MetaInstanceOf` is instantiated to a VSD instance and value `MetaProperties` are loaded. Composite references are processed recursively to build up the object hierarchy. In a downstream secondary pass, non-composite references are resolved. Document-external references are resolved by opening the corresponding XML file, loading the referenced portion and attaching it to the object hierarchy.

Saving the contents of the VSD to an XML document (marshalling) also works recursively, beginning at the root instance. Each instance’s `MetaInstanceOf` is retrieved and using the schema mapping an appropriate XML element is created. For each `MetaProperty`, XML child elements are added, which either contain simple values, further child elements representing target instances for composite references, xlink attribute values for internal non-composite references, or xlinks to external sub model elements.

Computer Aided Satellite Design (CASD)

The automatic construction of mission specific satellite configurations from modular building blocks is one goal of the iBOSS project. Concerning the high complexity and huge variety of modular and reconfigurable satellites, we developed the Computer Aided Satellite Design (CASD) tool chain (Göllner et al. 2012). The CASD tool enables operators and planners of new satellites to automatically generate satellite configurations from a standardized catalog of building blocks. The CASD synthesis process itself is internally divided into several steps to reduce the complexity of the overall optimization problem. Data has to be transferred and exchanged between the individual stages shown in Figure 5.
In the first step, mission specific parameters and resource specifications are used to select the required building blocks from the previously defined catalog. The outcome of the module selection is an XML file containing a list of selected building blocks referencing the catalog. The reasoning step uses this list of selected building blocks to infer rules and constraints for their placement. In the last step, the module arrangement uses these constraints to generate and optimize the satellite configuration consisting of the building blocks. The result of this optimization is a set of best fitting satellite configurations, which are written to separate XML files. With this representation, it is possible to easily exchange satellite configurations between all users of the overall tool chain. Furthermore, these satellite configurations are used to generate reconfiguration plans to describe the transformation from one configuration to another. These plans are stored in an XML file as well (Rühl et al. 2014).

For intuitive and simple handling of the XML data binding, we use the open-source and cross-platform tool CodeSynthesis XSD (Code Synthesis Tools CC 2014). It is a data binding compiler that generates C++ classes from XML Schema Definitions. The process of generating and compiling the C++ classes is integrated into our cmake based build framework. The generated data structure can be used to load, verify and save XML files complying to the XSD.

Automated Model Generation & Parameterization

The development of a modular satellite is in fact a variant of an optimization task: A module catalog has to be optimized until it fulfills the requirements of certain reference missions. This task is very complex and can only be done by iteratively building and testing satellites with different modules. As this process would take too much time when using real hardware, a sophisticated simulation is required, which can automate the task.

The simulation needs to be run with a large list of parameters for the configuration of the whole compound of the satellite as well as the building blocks themselves. The XML database contains these parameters ordered in three catalogs of modules, components and satellites.

The simulation model is based on the model description language Modelica. This language allows for a systematic and hierarchical integration of simulation models making the simulation architecture very modular. Figure 6 shows the model hierarchy used in the project. A library of components for subsystems of spacecraft like EPS (Electronic Power Subsystem), AOCS (Attitude and Orbit Control Subsystem), TCS (Thermal Control Subsystem) and DHS (Data Handling Subsystem) has been defined for this purpose in terms of Modelica packages.

Using the XML files, simulation models can now be generated automatically: First, a build tool parametrizes the models from the library with the information from the XML files and then stitches them together to form individual modules (for example, a model for a reaction-wheel module). In a second step, a connect tool analyzes the structure of a module compound and connects the individual modules accordingly to form a satellite. In a last step, the module is automatically compiled using a Modelica compiler.

(De-)serialization of the database between Python and XML is based on standard libraries (xml.etree.ElementTree) provided with Python 3.4. We do not use XSD-based generation tools (generateDS, PyXB, pyxsd), but wrote our own object-relational mapper (ORM) making extensive use of Python’s dynamically typed nature. XSD is mainly used for validity checks with our project partners. With this approach, we have more flexibility embedding the database into our own development chain.

**USE CASE 2: STANDARD MODELING LANGUAGES**

As a second use case, the modeling languages X3D and AutomationML were integrated into the simulation system VEROSIM. To apply an XML data binding approach, the basic modeling conventions of the respective XSDs have to be
identified. Subsequently, they have to be mapped to VSD’s modeling primitives, i.e., MetaInstance, MetaProperty, and MetaTypeVal. Given this general mapping, a corresponding data binding approach as presented above can be utilized to adopt the schema in the VSD and to load data from (and save data to) schema-compliant XML files.

After unmarshalling an instance document to the VSD, its contents are available to the components of the simulation system. However, the semantics or functionality of these objects are still unknown. For example, X3D or COLLADA geometry representations cannot immediately be interpreted by the simulation system’s render engine. Following (Hoppen et al. 2012), a functional data synchronization component can be developed to translate such representations online to the simulation system’s native format.

X3D

Following the analysis of the X3D format in (Schwellenbach 2014), abstract X3D node types are represented by global abstract XSD complex types. In turn, concrete X3D node types are represented by global XSD element definitions with an embedded, anonymous complex type to realize the content model and inheritance. Inheritance is modeled using XSD complex type’s extension mechanism. X3D fields representing simple (i.e., non-node) values are represented by XML attributes with special XSD simple types derived by restriction from build-in basic types (e.g., SFBool derived from xs:boolean). X3D fields that contain other nodes are modeled as XSD child elements referencing the corresponding node’s global element definition. XSD groups and attribute groups are used to model recurring content models and reference them from respective complex type content models (mostly sequences). Finally, the aforementioned DEF/USE statements are modeled using XSD ID/IDREF attributes that are members of every base complex type and, thus, of every type and global element. This allows each and every complex child element to become a reference to another existing element of the same type that was declared somewhere else using a DEF statement.

These modeling concepts are then appropriately mapped to the VSD meta system. Using these mappings, the X3D schema and corresponding data can be loaded into the VSD. Based thereupon, a functional data synchronization approach was developed for a downstream translation of geometry representations, transformations, materials, and texture definitions. An example is given in Figure 7.

AutomationML

As for X3D, the first step for an XML data binding approach for AutomationML is an analysis of the modeling conventions within the corresponding XML Schema Definitions.

In (Konieczny 2014), this is conducted with a focus on the CAEX root schema.

In general, CAEX uses a single global element definition CAEXFile with an anonymous complex type as an entry point to every document. Alongside, some basic global XSD complex types are defined. Within CAEXFile’s complex type, local elements with anonymous or named complex types are interleaved in a “Russian-doll” design. Besides, local elements as well as attributes with simple types are used to model non-complex properties. There are no abstract elements or type definitions in the CAEX XSD.

Again, these modeling concepts are mapped to appropriate VSD structures. Using these mappings, the CAEX schema can be adopted within the VSD and corresponding instance documents can be unmarshalled.

CONCLUSION AND OUTLOOK

We present an approach how XML-based modeling can be used to provide a common language for distributed teams working on or with 3D simulation applications. Depending on the application’s requirements, a customized or a standardized modeling language can be more advantageous. Furthermore, we show how XML data binding techniques can facilitate the integration of such a language into the different tools in heterogeneous teams. In particular, we illustrate how an XML Schema Definition for a customized language can be build up or how a standardized schema has to be analyzed to allow for a reasonable mapping to object-oriented structures. Altogether, this provides an excellent collaboration basis for distributed interdisciplinary research teams to cope with similar problems of data modeling and exchange.

Our aim is to develop standards for the integrated development of complex systems with 3D simulation techniques. We intend to support the whole life cycle of complex systems with electronic media, starting from system specification to design up to operation and maintenance. As the system spec-
ification is the first step that forms the basis, we currently work on a consistent evolution of a system design process starting from the specification. This evolution naturally contains iterations and parallel development, such that a consistent data management is a challenge. To this end, we want to combine the presented data structures for simulation with SysML for specification.

Acknowledgements

Parts of this work were developed in the context of the research projects iBOSS-2 and INVIRTES. Supported by the German Aerospace Center (DLR) with funds of the German Federal Ministry of Economics and Technology (BMWI), support codes 50 RA 1203 (iBOSS-2) and 50 RA 1306 (INVIRTES).

REFERENCES


