

Importing XMLSchema datatypes into TTCN-3*

D.-M. Jeaca, G. Din, A. Rennoch
Fraunhofer FOKUS, TIP
Kaiserin-Augusta-Allee 31, 10589 Berlin, Germany
www.fokus.fraunhofer.de/tip

Abstract:

There are various Systems under Test that describe the information to be exchanged using XML-based data type definitions (e.g. SOAP [9]). In this contribution we elaborate an approach to support the integration of such information in the test system. In particular the mapping rules of XML data types, i.e. XMLSchema definitions, to the Testing and Test Control Notation (TTCN-3) [4] and a tool implementation for this task are presented. Furthermore the difficulties and limits with such approaches are discussed in a case study on the MOST functional catalog.

1 Introduction

As the need for communication has grown a lot in the last decades, also the need of structuring the information went up with it. Information needed to be structured in a standard manner, for an easier way of creating, sending and reading it. So in 1996 XML (eXtensible Markup Language) was created by a group of SGML experts, sponsored by the World Wide Web Consortium. In 1998, Version 1.0 of XML was approved by W3C, and became a standard, that nowadays is the base of a lot of communication protocols. XML is a flexible tool for structuring data. Information from any vocabulary and in any structures can be restructured in an easy way, using XML. The base element in XML meta-language is the "tag". It is a kind of a label for the information that it encapsulates. XML language has a vocabulary, and also a grammar, given by the hierarchy of the tags.

In the context of automated systems testing it is necessary to consider available data type definitions coming with the System under Test (SUT) as much as possible during the test development phase to avoid any superfluous re-definition of such data structures for test definition.

* This work is partly sponsored by the ITEA project TT-Medal. <http://www.tt-medal.org/>

Among the broad variability of test description techniques we follow the only international standardized approach and use the Testing and Test Control Notation TTCN-3 [4] due to its wide applicability, acceptance and available tool support. One of the advantages of TTCN-3 is the availability of both standardized open runtime and control interfaces (TRI and TCI) that enable an exchange of tool components between different test tool providers.

After a short introduction on XMLSchema, TTCN-3 and the import mechanism this paper provides an overview on the mapping rules and a prototype implementation for an automatic translation of XSD to TTCN-3. For implementation issues we follow the TTthree tool chain [6] because it is comprehensive w.r.t. management and logging facilities and has a good portability.

Since this mapping has been applied on testing MOST-based systems we shortly introduce the major technical characteristics of MOST: The MOST functional catalog Fcat that provides a full specification of the devices and its functions within a MOST network will be considered in Section 5 w.r.t. its application within a TTCN-3 test suite. Details on mapping results and further approaches will be reported, too. Finally we give some conclusions from our work.

2 Data types integration in TTCN-3

TTCN-3 is a flexible, text base language, used for testing reactive systems that use any communication interfaces. The areas of application that can use TTCN-3 are: protocol testing (mobile and Internet protocols), service testing, module testing, CORBA based platforms testing, API testing etc. It is used for conformance testing, but also for other kinds of testing, like interoperability, robustness, regression, system and integration testing.

The third edition of TTCN (TTCN-3) has been developed to address the needs of testing technologies in the telecom and datacom domain. The extensions that it has to the previous standards address e.g. testing synchronous communication, new flexible matching mechanisms to compare the reactions of the SUT with the expected range of values, or the ability to specify encoding information.

As illustrated in Figure 1 TTCN-3 allows the import of types and components from other languages, e.g. ASN.1, IDL [3] and now XMLSchema [2]. This import mechanism makes TTCN-3 more flexible and extensible in the sense that test developers do not have to specify data types already available in other languages but can reference and use them.

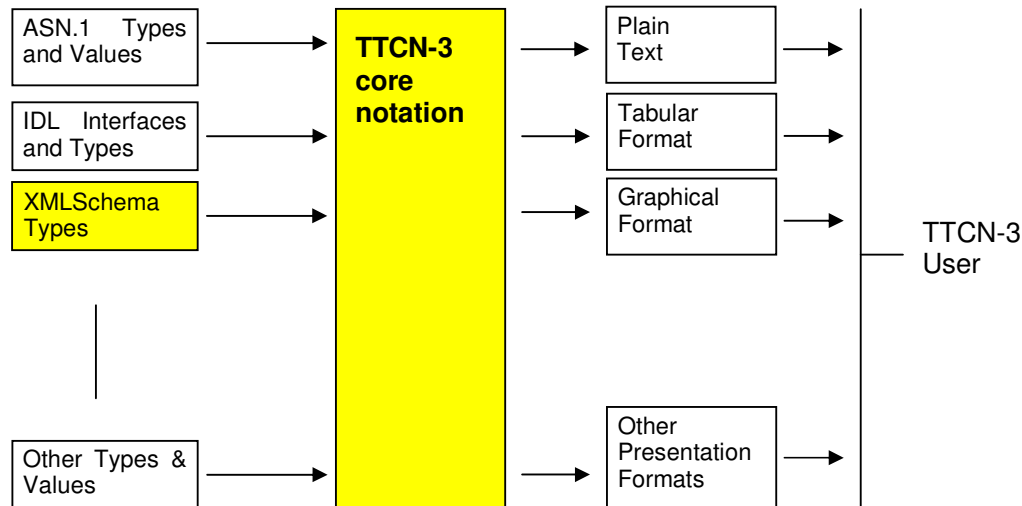


Figure 1: TTCN-3 language and format mappings

Furthermore Figure 1 shows that TTCN-3 encompasses different presentations formats in addition to the core notation, e.g.:

- Tabular presentation format uses collections of tables to define a TTCN-3 module.
- Graphical presentation format uses Message Charts manner for describing visually the module behaviour.

In the latter case, i.e. the graphical format for TTCN-3 (GFT), there are special diagrams for basic elements, for example: *test case* diagrams, *control* diagrams, *function* diagrams, etc. But the core language can be used independently of the presentation format. Thus it is sufficient that the mapping from XMLSchema to TTCN-3 focus on the core notation of TTCN-3, only.

3 The mapping rules

In [1] a generic approach for XML to TTCN-3 has been investigated. The main issues of this approach is the mapping of the XML built-in types to TTCN-3 types (e.g. decimal to TTCN-3 float), the consideration of XML facets (e.g. maxInclusive leads to TTCN-3 value restrictions), and the generation of structured TTCN-3 types from XML ComplexType specifications (e.g. choice is translated to TTCN-3 union). An auxiliary TTCN-3 module has been introduced for the mapping of all built-in types without facets to support short references within TTCN-3.

The structure of the mapping follows somehow the structure of XML Schema. First we look into the built-in datatypes, and afterwards into the components that the language offers.

Built-in datatypes are structured into primitive ones, and derived ones. The latter are derived from the primitive ones, by means of restrictions, like: length, size, list, range etc. These restrictions are called facets. For every simpleType, primitive or derived, facets can be applied, and a new type will be available. So for every built-in type there is a list of possible facets that can be applied to it, and depending on the facet, the correspondence in ttcn-3 is different and included. Every built-in type, without any facets, is mapped with its built-in name, in a module called XSDAUX. And whenever a new simpleType is defined, with the base type a built-in one, it will be mapped using dot notation.

Example:

```
<xs:simpleType name="I">
  <restriction base="xs:integer"/>
</xs:simpleType>
```

Becomes:

```
type XSDAUX.integerXSD I;
```

Every built-in type is first mapped into the TTCN-3 definition of a simple type, and afterwards for the definition of the type, with every available facet. After mapping the basic layer of XML Schema, i.e. the built-in types, the mapping of the wrapping structures follows. For that, every structure that can appear, globally or not, will have a corresponding mapping into TTCN-3.

SimpleType components are used to define new simple types, by three means: restricting a built-in type by applying a facet to it (as discussed in the previous section), building lists or unions of other simple types. SimpleTypes can be defined globally, which means the parent is <schema>, and the <name> attribute is mandatory, and they will be mapped to the new TTCN-3 type using that name. And so they can be reused in other definitions. Or they can be defined locally, i.e. the <name> attribute does not appear, so they will be mapped with an automatic generated name, but they will not be reused in other definitions. Here we will give just another small example:

```
<simpleType name='eRestriction'>
  <restriction base='string'>
    <pattern value='\d{3}-[A-Z]{2}'/>
  </restriction>
</simpleType>
```

Becomes:

```
type XSDAUX.string eRestriction (pattern "[0-9]#(3)-[A-Z]#(2)");
```

The complexType is used for creating new types that contain other elements and attributes. This is in contrast with the simpleTypes that cannot contain attributes. Just like simpleTypes, complexTypes can be defined globally, which means the possible parents are: <schema> and <redefine>. When they are defined globally, the "name"

attribute is mandatory, so the new types will be mapped under the given value of that attribute. And they can be defined locally, in which case the name attribute cannot appear, they will be mapped using an automatic generated name, but they will not be used in other complexType definitions. In other words, they cannot be referenced from other definitions, as their purpose was locally.

For the mapping, the idea is to take separately every child that it can have, and assign the corresponding TTCN-3 code. The children of complexType are:

- annotation?,
- simpleContent | complexContent
- ((group | all | choice | sequence)?,((attribute | attributeGroup)*,anyAttribute?)),

where "?" means zero or one time, "*" means zero or many times and "," can be translated to "followed by".

4 Implementation

There is also a prototype implementation available that has been integrated into a commercial TTCN-3 compiler TTthree [6]. The implementation allows both an implicit mapping of XML structures into an internal tree representation of the XML definitions and also the explicit translation into TTCN-3 core notation syntax due to the application of some TTthree printing features in a second step.

The translation from XMLSchema to TTCN-3 could have been a stand-alone tool, but it is needed for the import handle into TTCN-3, so it has been included in the TTthree compiler. For the integration in the TTthree compiler, the mapping is divided in two parts: "implicit" mapping, that translates XMLSchema to the TDOM (Typed Document Object Module) structure, and "explicit" mapping that translates from the TDOM structures to TTCN-3 syntax. The first mapping is obtained from the XMLSchema parser that will be defined below, and for the second one is used another tool, also integrated in the TTthree compiler, Syntax2Template.java. The TTthree tool is a Java application that comes with its own Java Runtime Environment (JRE 1.3.1) and Java compiler (IBM's Java compiler Jikes v. 1.16).

Creating the TDOM is one step in the import of XSD types into TTCN-3. The new types are assigned to the tree, and any mistake in the compilation will be noticed if events are not sent in the right order. But to verify that the translation from XMLSchema to TTCN-3 was correct, it would be too clumsy just to watch the tree. So a tool (Syntax2Template.java) has been used, for translating from TDOM to TTCN-3 syntax. And the verification of the translation now can be done really easy. So the short version is: from XMLSchema to TTCN-3 TDOM, in the TTthree compiler, and from TDOM to TTCN-3 syntax, using Syntax2Template.

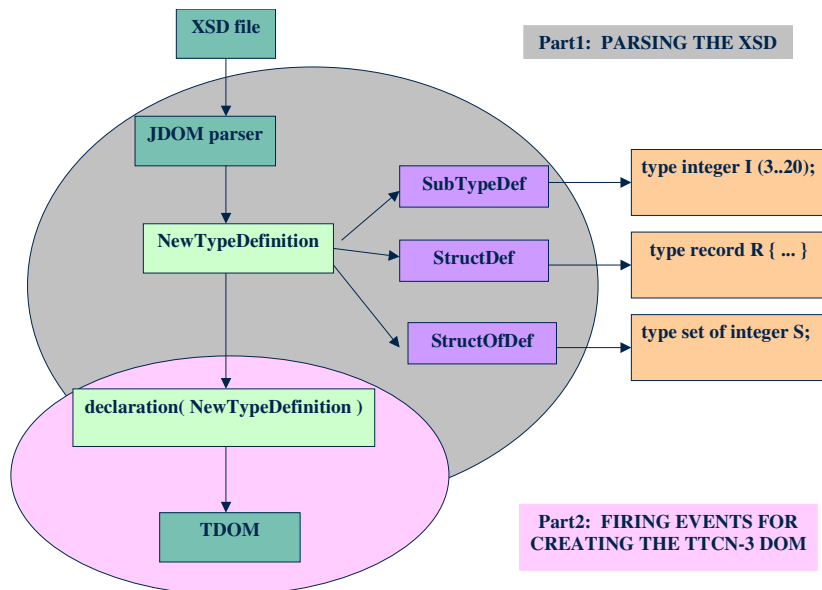


Figure 2: From Parsing to firing Events

5 Case Study: MOST functional catalog

5.1 MOST systems

MOST and CAN busses are accepted standards between European car developers. A MOST-based system makes it possible to transfer real time data (CD audio, CD video), control commands for other MOST devices (media player, active loudspeakers, MPEG decoders) and packet based data over one bus. The devices on the bus are independent components, which are able to exchange messages and data. The communication between the various telematics in a car components must be explicitly specified and verified.

Every device on the MOST-Bus is provided with an address (*Device-Address*), with which it can be uniquely identified. Several functions are implemented in one device and grouped into so called function blocks. Every function block has a distinctive identifier (*FunctionBlock-ID*). The definitions concerning the identifier are part of the specification and are stored in an XML data base. Every function block implements functions and properties which, in turn, can be called by their identification code (*Function/Property ID*). Every function/property can be called with different access attributes (*Operation Type*). For functions, these are SET, GET, SETGET, GETINTERFACE, and for properties additionally INC, DEC. This way, function

executions can be initiated, the transfer of the result can be requested or the interface version can be queried. For properties, operations for setting, collecting, incrementing or decrementing are available.

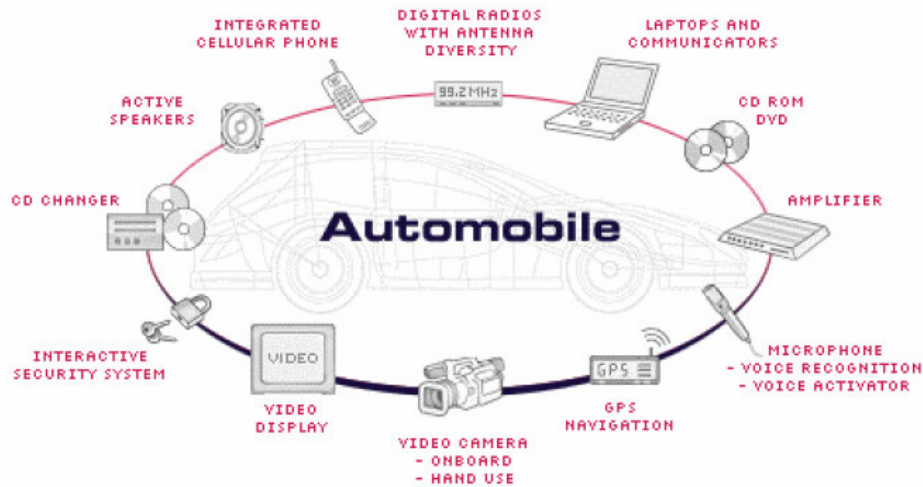


Figure 3: Automotive telematics systems

5.2 MOST Fcat

The MOST functional catalog (Fcat) comprises the definitions of MOST devices and functions. Each function may have different operation types with a related parameter list. Beside the English text version and different printed tables on the catalog the MOST cooperation provides two machine readable documents on Fcat: a DTD file on the structure of function blocks and function interfaces (fcat.dtd) and a corresponding data file most.xml which gives information on the concrete function signatures and parameter types.

It is important to be aware about the distribution of the Fcat information among the two files. The type definition file provides a tree structure describing the set of MOST functional blocks and functions on a high abstraction level, i.e. it introduces the scope of information related to all MOST functions without addressing any particular function (identifier, parameters etc.). Beside some general elements in this tree about the identification and version of a function the main structure address the possibility of function properties and methods. Especially the latter introduces e.g. different operation types for method commands and reports. Furthermore it provides the list of allowed parameter types in MOST but not the relation of parameters to functions.

Function and parameter names and its textual descriptions as well as the details on the parameter types (e.g. value scope restrictions) are part of the XML file only. The XML file contains the information on any parameter position within a MOST function. Furthermore it provides some information for the protocol data coding (e.g. FBlockID, FunctionID).

It is obvious that testing MOST-based communication has to consider the data structures given in the DTD and XML files. It should be possible to address the Fcat contents within the TTCN-3 test suite behaviour and data templates. Therefore an automatic translation of the MOST Fcat into TTCN-3 is required

5.3 Mapping of the XMLSchema in MOST

To get the most flexibility a general approach for the MOST Fcat transformation has been selected: First, the DTD definition will be translated into the popular XML schema definition (XSD) format and secondly a generic XSD to TTCN-3 translation need to be performed. We've applied both, the freely available XML DTD to XML Schema Converter from Syntext [5] and our XSD to TTCN-3 translator [1]. Due to the nature of the MOST Fcat contents the resulting TTCN-3 data types have been quite complex for human reading and a modified approach for the integration and usage of the MOST definitions will be proposed.

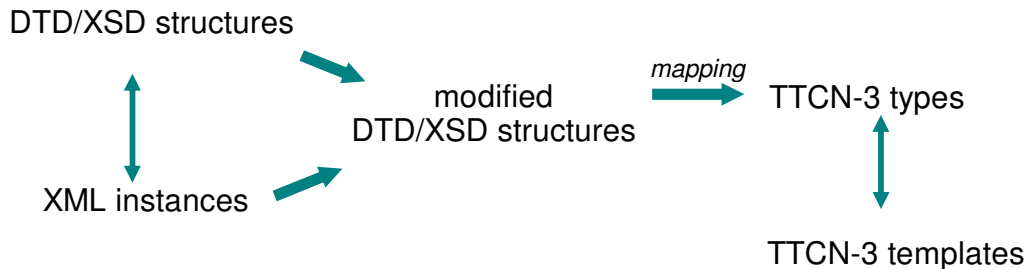


Figure 4: MOST Mapping approach

Following the approach illustrated in Figure 4 three steps have to be considered: (a) some restriction of the DTD/XSD structure to the those data elements that are observable during testing, (b) an integration of XML data into the DTD/XSD structure definition, and (c) the mapping to TTCN-3 data types that will be used for the declaration of data templates.

In the first phase we've done the modification of the DTD/XSD structures manually instead of applying some automated techniques like XSLT. It seems sufficient to use some XSD definition for the MOST data types like the following:

```

<xs:element name="ConnectionMaster">
<xs:complexType> <xs:choice>
  <xs:element ref="ConnectionMaster_BuildSyncConnection" />

```

```

        <!-- further functions provided by ConnectionMaster -->
</xs:choice> </xs:complexType>
</xs:element>

<xs:element name="ConnectionMaster_BuildSyncConnection">
<xs:complexType> <xs:choice>
    <xs:element ref="StartResult" />
    <!-- further operation types -->
</xs:choice> </xs:complexType>
</xs:element>

<xs:element name="StartResult">
<xs:complexType> <xs:sequence>
    <xs:element ref="Source" />
    <xs:element ref="Sink" />
</xs:sequence> </xs:complexType>
</xs:element>

```

Listing 1: Transformed MOST XSD definitions

The resulting TTCN-3 data types has been produced with the XSD to TTCN-3 translator and could be referenced in test cases data templates:

```

type union ConnectionMaster__Element {
    ConnectionMaster_BuildSyncConnection__Element
    ConnectionMaster_BuildSyncConnection
}
type union ConnectionMaster_BuildSyncConnection__Element {
    StartResult__Element StartResult
}
type record StartResult__Element {
    Source__Element Source,
    Sink__Element Sink
}

template ConnectionMaster_BuildSyncConnection__Element CM_BSC_SR :=
{StartResult:= {
    Source := {SourceFBlock:=1, SourceInstID:=2, SourceNr:=3}
    Sink := {SinkFBlock:=1, SinkInstID:=2, SinkNr:=3}
}
}

```

Listing 2: TTCN-3 data types and templates

6 Conclusions and future work

This paper presents the enhanced concepts, a tool implementation and experiences w.r.t. the import and use of XMLSchema for TTCN-3 test developments. As described in details a pure XMLSchema to TTCN-3 is not sufficient since superfluous structures may be included or important structure information are given in related XML files only.

Furthermore there are some additional ideas for future work: The next working issue is the parsing of XML instances. After getting the datatypes from the XMLSchema files, the actual values of these types may be retrieved for the generation of some TTCN-3

templates. One approach that can be thought of for the implementation of this task may use JAXB (Java API for XML Binding), which binds XML structures with Java classes, from an XMLSchema file, and then the objects are obtained from the XML instances. It is obvious that developing templates that applies forbidden data values (in the “invalid” test behaviour group) still needs additional efforts.

Another issue address the implementation of the test system codec. In order to send real data over a real communication port, or call a real world method on a SUT port, encoding functionality has to be provided to the TTCN-3 runtime behavior. On the other hand incoming data has to be decoded into a TTCN-3 value, to allow matching against TTCN-3 template definitions. Due to the import of XMLSchema datatypes it is likely that the test system exchanges XML based data with the SUT. As described in [1] one prominent sample is the SOAP protocol that includes a mechanism for changing structured and typed information between peers, using XML. In this situation the availability of a Codec implementation that corresponds to the TTCN-3 data types resulting from the XSD translation is strongly desired. The automatic production of such Codec is considered as an important issue.

References

- [1] D.-M. Jeaca: XMLSchema to TTCN-3 Mapping. Diploma thesis. Politehnica University Bucharest, September 2004.
- [2] G. Din: The XML to TTCN-3 mapping. TTCN-3 user conference. Sophia Antipolis, May 2004. <http://www.ttcn-3.org/program.htm>
- [3] M. Ebener: The IDL to TTCN-3 mapping. TTCN-3 user conference. May 2004.
- [4] ETSI ES 201873, Testing and Test Control Notation (TTCN-3), Sophia Antipolis, February 2003. <http://www.etsi.org/ptcc/ptccttcn3.htm>
- [5] Syntext dtd2xs v2.0 user's guide, Syntext Inc. 2003, <http://www.syntext.com/products/dtd2xs/doc/index.html>
- [6] TestingTechnologies IST GmbH: TTCN-3 compiler TTthree. <http://www.testingtech.de/products/ttthree.php>
- [7] MOST cooperation: <http://www.mostnet.de/downloads/Specifications/MOST>
- [8] World Wide Web Consortium: XML Schema, W3C Recommendations, May 2001. <http://www.w3.org/XML/Schema>
- [9] World Wide Web Consortium: SOAP documentation. W3C Recommendation. <http://www.w3.org/TR/soap>