SHARED DATA TYPES FOR OSRA AND TASTE

- MBSE 2020 -

28-29 September 2020

Noordwijk, Netherlands

Jan Sommer[(1)], Andreas Gerndt[(1)], Daniel Lüdtke[(1)],

[(1)]DLR (German Aerospace Center), Institute for Software Technology
Lilienthalplatz 7, 38108 Braunschweig, Germany
Email: jan.sommer@dlr.de
Email: andreas.gerndt@dlr.de
Email: daniel.luedtke@dlr.de

ABSTRACT

Demand for more complex on-board functionality for future spacecraft continues to rise, increasing the burden on often small software teams. To handle the complexity of modern on-board software, model-driven methodologies can help to capture the overall architecture and design of the software. In a later step, they also allow auto-generating source code and documentation artifacts from the model, thereby relieving software developers from monotonous tasks.

In order to support model-based software development, the European Space Agency (ESA) provides *The Assert Set of Tools for Engineering* (TASTE) and the *On-Board Software Reference Architecture* (OSRA). Both share some common design concepts like separation of concerns, component-based modeling and graphical tooling for the design tasks. However, OSRA targets mostly the design of spacecraft on-board software. At the same time, it leaves the concrete implementation of the code generators to the entity using OSRA. TASTE, on the other hand, provides a more generic framework, includes code generators for the C and Ada language and has also been applied in robotics applications as well. Unfortunately, the interworking between the two frameworks lacks a mechanism to exchange data easily without duplicating the data type information.

In complex software projects, data is exchanged by a plethora of software modules, potentially developed by different software teams. To ensure safe data exchange inside a single as well as across many different modules, essentially three basic problems need to be solved: First, there has to be a common source defining the data types, ideally with the option to define constraints. Secondly, there needs to be some way to determine if a value is valid or not. Finally, there needs to be a shared understanding about how to encode values of these data types. TASTE solves the first problem by describing its data types through the language agnostic *Abstract Syntax Notation One* (ASN.1) notation. The second one is addressed by generating test routines for validity checks during runtime. The last problem is solved by implementing encoding rules of ASN.1 or by using a custom description for the encoding of data types using TASTE's *ASN.1 Control Notation* (ACN). OSRA currently only solves the first problem through its graphical data type editor.

Our goal is to allow the exchange of data between software developed with both tools without the need for manual interventions. In the modeling phase this is mainly achieved by adding ASN.1 capabilities to OSRA: data exchange is carried out between OSRA and TASTE applications by using the same ASN.1 representation for the data types. In OSRA, data types are modeled graphically and are part of the overall model which is based on the *Eclipse Modeling Framework* (EMF). We added plugins which auto-generate ASN.1 data types from this internal representation, allowing TASTE to use them. The code-generators are implemented using the xtext/xtend framework. To make the integration bi-directional, we also implemented a basic set of the ASN.1 grammar with the xtext programming framework. It provides an editor with syntax highlighting for ASN.1, but more importantly, it allows to parse existing ASN.1 data type descriptions, e.g. from TASTE, and register them as *external types* in the OSRA model. In the end, all available data types, the ones generated by the graphical editor as well as textual ones, are captured in this model. With this approach, the first problem is solved for OSRA and TASTE by the same way using an ASN.1 notation to describe data types.

For solving the second problem for OSRA, we developed an alternative concrete implementation for the data types generated from the ASN.1 type description. In contrast to TASTE's *asn1scc* generator, the target language is not C but modern C++ using features introduced in the new standards C++11 to C++17. DLR had successful missions with on-board software written in a subset of the C++ language, such as the TET-1 and Eu:CROPIS satellite missions, the MAIUS-1 sounding rocket mission, and continues this path in the upcoming ReFEx re-entry vehicle. With today's wide

availability of compilers with support for modern C++ standards, even for embedded targets, it is now possible to leverage the newly introduced features. At the same time, common constraints for spacecraft on-board software, e.g. the avoidance of dynamic memory allocation, are kept.

The much more powerful templating system of modern C++ allows developing templated meta-classes for the common base types with build-in constraint checking. For example, numerical types can check the values based on their ranges and choice type variables can track their active field. In general, constraint compatibility is checked during type conversion. Template meta-programming techniques, i.e. variadic templates, type traits, constant expressions and static assertions, play a major role in the implementation. They direct the C++ compiler to generate the type checking for a concrete type instance. Since template resolution in C++ needs to be carried out at compile time, also many checks will produce a compile time error if certain conditions are not met, e.g. assigning numerical types with incompatible ranges, thereby directly prohibiting the introduction of possibly dangerous code into the source tree.

In situations where compile-time checks are not possible, runtime checks are carried out. They are triggered automatically by the type variable itself when its value changes, freeing the developer from remembering it manually and avoiding code clutter. Finally, with the complexity for value checking captured in the type meta-classes, the C++ code produced by the code generator from the ASN.1 input is comparably simple. This increases maintainability for the code generator. For numerical types, this can often mean simply a one line *using* statement. For structured types, only the mapping between field name and type needs to be generated. This approach leads to a clean API for the users of the type system.

To solve the problem with shared encoding rules between TASTE and OSRA, we added a prototype implementation for a serialization mechanism to the aforementioned type system. It aims to be compatible with the ASN.1/ACN encoded binary streams of TASTE. With this step, software developed with both frameworks can now exchange data based on the same data type description. TASTE also supports additionally several ASN.1 encoding rules. This is currently not the case for our data type framework. ACN was chosen for the first implementation since it allows the definition of encoding rules which makes it applicable for existing communication protocols. However, the framework is flexible enough to add further encoding rules in the future. The *Basic Encoding Rules* (BER) and *Packed Encoding Rules* (PER) of ASN.1 are good candidates to add next.

This work shows the additions to the OSRA infrastructure in order to allow the exchange of data between OSRA and TASTE based on the same data type descriptions in ASN.1. This includes enabling OSRA to read and write ASN.1 data type descriptions, the implementation of the data types in modern C++ and the serialization of the data types into an encoded binary compatible to TASTE. Some first results about the exchange of data between both frameworks are presented in this work as well. Of course, the work presented here is not only useful for the data exchange between both frameworks but also builds the basic type system on which our OSRA based code-generation can build upon in the future.