Model Driven Language Framework to Automate Command and Data Handling Code Generation

Meenakshi Deshmukh, Benjamin Weps
German Aerospace Center (DLR)
Simulation and Software Technology
Lilienthalplatz 7, 38108, Braunschweig, Germany
meenakshi.deshmukh, benjamin.weps@dlr.de

Pedro Isidro
Instituto Superior Técnico
Department of Mechanical Engineering
Avenida Rovisco Pais 1, 1049-001 Lisboa, Portugal
pedro.isidro@tecnico.ulisboa.pt

Andreas Gerndt
German Aerospace Center (DLR)
Simulation and Software Technology
Lilienthalplatz 7, 38108, Braunschweig, Germany
andreas.gerndt@dlr.de

Abstract—**On-board computer software (OBSW) is an integral part of every space mission. It has been continuously growing in size and complexity. The insufficient level of automation in the development process of such software leads to low software reusability and drives up the costs. This paper presents a generic approach to describe and model the on-board software in terms of data that is processed by it. Domain Specific Language (DSL) based framework is developed using which provides a DSL editor, a model validator, and a code generator. Using the framework, a system data model is created. The C++ code is generated from it which is then customized to implement low-level behavior. As a proof of concept, the telecommand handling functionality of OBSW is developed to prove the feasibility of applying the solution to the whole system. Based on the analysis conducted on the source code of the TET-1 satellite of the German Aerospace Center (DLR), a DSL is designed and implemented. The resulting DSL-based framework is tested with an example model and target code customization, showing its ease of use and proving that it behaves as expected.**

TABLE OF CONTENTS

1. INTRODUCTION

The size and complexity of On-board Software (OBSW) of spacecrafts is steadily increasing. It contains hundreds of thousands of lines of code responsible for the whole functionality of the spacecraft. Generally OBSW consists of software from different subsystems like Thermal Regulation, Attitude and Orbit Control (AOCS), Guidance and Navigation Control (GNC), Power Distribution Unit (PDU) etc as shown in Figure 1.

Despite of different mission objectives for each spacecraft, functionalities like handling telecommands, serializing custom sensor data for telemetry reports or managing internal distribution of data are usually common for all spacecraft missions. Traditional software development methods based on manual coding are adding considerable cost for developing such repetitive tasks and therefore no longer suitable. For example consider a Command and Data Handling (CDH) subsystem which is a vital part of on-board software of every spacecraft system. This subsystem is responsible to process the data which has to be exchanged between the ground-station and the spacecraft. As for every communication, a common protocol has to be defined which describes the structure of the transmitted data. For this purpose there are already standards available (e.g., ECSS PUS Standard [2], CCSDS [3]). In general these standards are very universal and have to be tailored to the specific mission where they are used in, due to the very broad requirements of each mission. In most of the projects this tailoring is done individually for each mission. Also for small projects, like sounding rocket missions, the overhead of a full-sized protocol would be too high and a custom protocol is preferred. In both cases the output of these processes are hardly reusable as they are most likely designed for one mission only. This circumstance brings up the problem, that one has to start all over to implement a quite similar functionality in different projects.

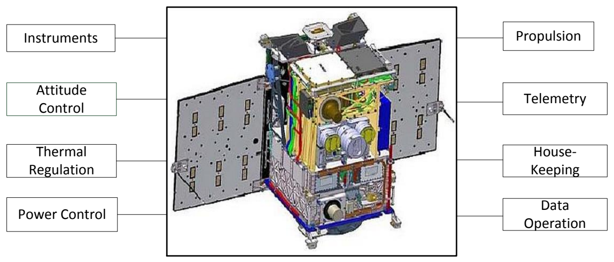


Figure 1. General Structure of Spacecraft's On-Board Software System (TET-1 Image Credit [1])

This low-level design and development has many drawbacks, which become increasingly significant as the complexity of the system grows. Firstly, having human developers carrying out repetitive low-level tasks is highly unproductive, since they are focusing on implementation details instead of design decisions. Secondly, humans are prone to make occasional errors, as opposed to computers, which only produce systematic errors which are easier to trace. Lastly, the fact that

978-1-4799-5380-6/15/$31.00 ©2015 IEEE.

software development is carried out at a low level leads to low re-usability.

The software usage of Attitude and Control Software (ACS) of DLR's BIRD mission was successfully applied for TET-1 mission [1], [4]. The reuse analysis shows approximately 74.7% of ACS code can be taken over unchanged. This reuse is possible because BIRD's ACS was developed keeping the idea of software reuse in mind. The significant results from this use-case has strongly made us to put more efforts in the direction of software reuse in case of on-board software.

One solution of this problem, which is presented in this paper, is to use a model-driven approach for defining the data to be transmitted. This approach has the big advantage that it enables clarity and comprehensibility of the whole subsystem while being flexible enough to match the mission requirements. This closes the gap between the already used standards and the mission specific data and reduces cost of reoccurring tasks. A model driven language framework is developed at the department of Simulations and Software Technology (SC) at German Aerospace Center (DLR) to streamline the software development process for Command and Data Handling. The framework provides a domain specific language (DSL) to model high level description of the data structures used in the spacecraft system. As a part of the framework design, requirements and implementations of previous missions were analyzed to identify commonly used data structures and protocols. Based on this, a generic model of data structures has been designed. This model is then used for generating code to support the development of the on-board data handling software.

The model provides higher level of abstraction which allows not only a quicker development of complex systems but also it provides an opportunity for an early verification of the system design. It has been shown that about 70% of faults are introduced early in the development process. And 80% of those caught only at the stage of integration testing or later where the cost of fixing them is much higher [5]. By validating the model, which is directly linked to the target system, many of these faults could be eliminated early in the development process. The model can also advance earlier phases of the spacecraft engineering process. When the data transmissions are modeled in the early phases, one can keep track of data usage and the data budget of the telecommunication.

The paper describes current initiatives taken in this direction in Section 2. Section 3 describes concept and scope of the language framework developed in order to automate OBSW software process. Section 4 provides detailed DSL implementation with respect to AOCS subsystem and CDH. Finally, the paper ends with a conclusion and insight to future work in Section 5.

2. State of the Art

First of all we want to differentiate between Model-Driven Software Development (MDSD) and Model-Based Software Development (MBSD). MBSD uses models merely as documentation artifacts [6]. On the other hand, MDSD puts models at the core of the development process. These models are then used to automatically generate platform-dependent designs, source code, tests, and even documentation [7]. This significantly increases not only software reuse but also allows the developer to abstract away the implementation details, which are not relevant to the problem domain.

There is also distinction between descriptive and prescriptive models. A descriptive model is mostly developed for communication and/or analysis of a system, while a prescriptive model is used to partially (or fully) automate the implementation of the system [8]. Therefore, when referring to models in the context of MDSD, one refers to prescriptive models which imply a higher level of detail. Such models are created to gain a more meaningful insight into the system, more reusable software, and better integration of design, implementation and testing [9].

With this background about MDSD, now we describe the different initiatives taken in the direction of modeling and code generation for spacecraft's on-board software development.

Taking inspiration from AUTomotive Open System ARchitecture (AUTOSAR) which is an open and standardized automotive software architecture, the initiative called Space AVionics Open Interface aRchitecture (SAVOIR) [10] has been started. The SAVOIR initiative is mainly promoted by the European Space Agency (ESA) to define comprehensive standards and processes for a better collaboration of the European Space community. One output of this initiative is the "Avionics Reference Architecture" which provides a common set of interfaces for spacecrafts. This description can be used as a starting point for developing a interface architecture which has the potential to get widely accepted.

As a part of the SAVOIR initiative, the so-called "SAVOIR-FAIRE" group is working on a software architecture for use inside the SAVIOR concept. One output of the working group is the COrDeT on-board software reference architecture [11]. COrDeT is following a component-oriented concept where the software is built out of replaceable building blocks. This concept is very flexible and enables the development of "plug-and-play" software.

Space Plug-and-play Avionics (SPA) focuses on using a plug-and-play approach to allow easy replacement of space avionics components. [12]. At the software side, the concept involves developing a middleware layer called Satellite Data Model (SDM) to which all components are linked to. This layer aims at promoting software reuse at the application level.

Other than the software architecture and interfaces, another key aspect of the command and data handling subsystem is the data structures that protocols used for communicating with the ground support environment. Current available standards for telemetry and telecommands are designed on the abstract level and have to be tailored to the mission requirements. The tailoring then consists in the definition in the actual data structures transmitted with these protocols. One method to define the structures in a consistent way is provided by the widely used Abstract Syntax Notation One (ASN.1) [13]. It provides a platform independent way to describe data structures. Out of this notation, the representations and serializing methods for different programming languages can be generated. Although ASN.1 provides a way to describe the data, that notation is designed for this purpose only and extending it for other applications and functionalities is not sophisticated.

The above mentioned approaches are going in the direction of MBSD. They cover modeling of only high level system and/or software architecture. The models are at the descriptive level. Since we want to achieve the automation of low level repetitive programming tasks, we need a prescriptive

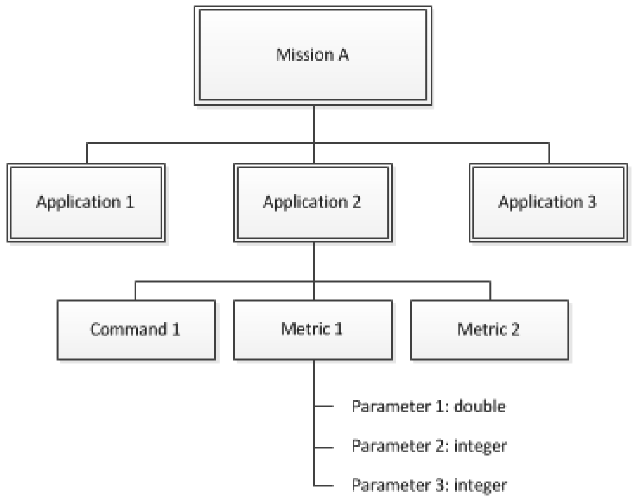


Figure 2. The hierarchy of the description model

way of modeling.

While modeling the on-board software in a prescriptive way, we came across the different options of modeling. For example, graphical or textual, general-purpose or domain-specific etc. We compared the possibilities and we preferred to use textual and domain-specific modeling for two reasons. First, we are addressing a special set of software here that is on-board software and therefore we need means to model domain specific features. Second, we want to model low level implementation details of a software. And with a textual model, we can provide such details in an extensible way.

For embedded applications, DSL-based projects have already been undertaken successfully [14]. It has also been shown that for real time requirements the usage of DSLs and Model-Driven-Development is also feaseable [15].

So we have decided to develop a domain specific language to model on-board software which will automate recurrent design patterns beyond the architecture.

3. Concept Development

The main concept of the Data Description Language (DDL) is to provide a data-centric model of the satellite system. This model defines the different structures which will be transmitted throughout the mission execution.

Hierarchy

The files from TET-1 mission were analyzed for structure, functionalities and behavior. The analysis of the TET-1 files follows a bottom-up paradigm, starting with the C++ classes corresponding to the command handlers. For the better understanding of command handlers and finding common patterns or features, several instances were taken and compared. Based on this analysis and from the experience of domain experts from previous missions, the generic mission data architecture is divided into four hierarchical layers, which is seen in figure 2.

The uppermost layer is the mission layer. It only contains one element: the root node. This unique node describes the general parameters for the mission. It also holds a set of all applications that will be used for this mission which are settled in the next lower level. Applications allow a grouping of data which have the same or a similar context. Each application holds a set of metrics and commands that it is able to process. An application can contain multiple internal components. The grouping of data can overlap, which means that for example data of a single sensor can be contained in more than one application and multiple applications can command actuators or internal procedures. The concept of application matches with the applications of the CCSDS standard. The next lower level contains the structures that make up a single metric or command of a certain application. This structure is self-containing and maps to one packet of the underlying transmission protocol. Every packets consists of an ordered list of basic data types. These fields are used as parameters for the Metric and Command packages. In conjunction with Metric packages the parameters contain the actual data from the sensors and the states of the system. As a part of a Command package the parameters define the data associated with this command.

This concept allows not only a description of the data which will be transmitted but also the interfaces to the different software components of the on-board software. This leads to a so-called data-centric approach for the on-board software development. Instead of the system bus concept for space systems described in the paper of Venigalla et al. [16], the data-centric approach puts the command and data handling component into the center of the architecture.

The advantage of this architecture is to have a single point where the definition of the data and interfaces is implemented, easing the effort to manage changes and additions to these definitions.

Scope

When examining the entire chain of data transmission and processing, as shown in Figure 3, the primary scope of the DDL is the data transmission between the spacecraft and the ground station.

The DDL provides a standard for the representing data for the transmission on the application level. This includes the high-level specification of the data packages for telemetry and telecommands, and the corresponding serialization mechanisms. As mentioned in the hierarchy section, the DDL is also able to describe the interfaces to the different applications used in the on-board computer. Therefore the scope of the DDL also extends to parts of the command and data handling component.

Figure 3 also shows the concept of the data-centric architecture. Here the command and data handling component is considered different to the remaining components of the on-board data processing. The dotted line in this figure shows the scope of the Data Description Language. The language mainly covers the serialization methods and datatypes. Additionally, stubs and interfaces for the Command and Data Handling subsystem can be defined, as the structure of all messages which will be handled is known.

Corresponding to the domain model in Figure 2, a semantic Data Description Model (DDM) is developed which is defined in Figure 4. One to one mapping between each level in both models can be easily seen.

Based on this semantic DSL model, a set of DSLs are

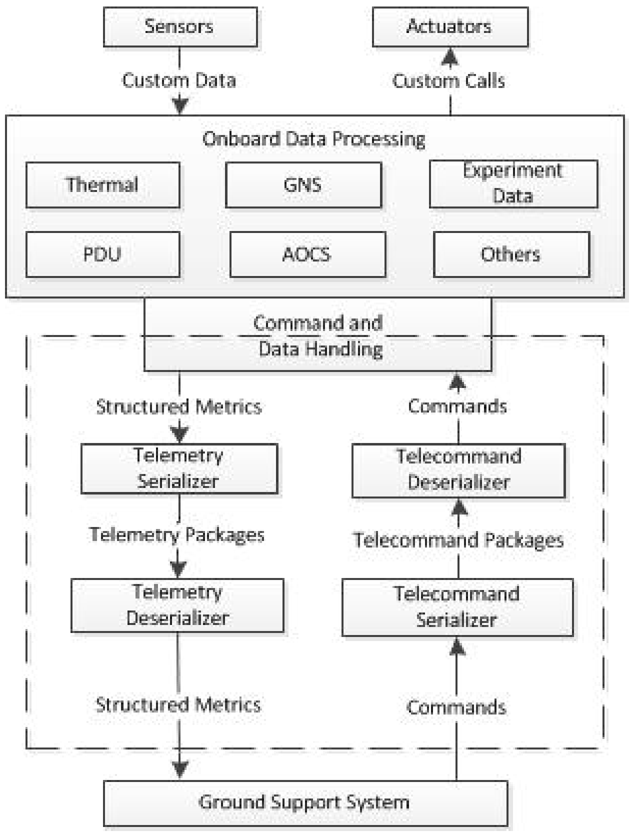


Figure 3. General Structure for Command and Data Handling

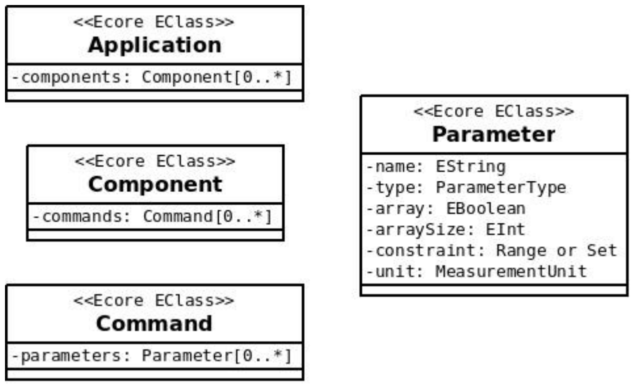


Figure 4. The semantic DSL model

designed to fulfill requirements stated in Section 1. The aim is to provide mission independent templates which will used in future missions to define mission dependent specifications. From these, mission specific code and tests will then be generated automatically, resulting in a reduced development overhead. The DSLs are defined in hierarchical manner with two levels. First level is the *Common Language* module. The global data types and common terminologies are defined in this DSL. The second level is the *Application Language* module. In this, the dsl provides a way to model a data structure required to define specifications of respective applications. As mentioned above, the scope of this paper is limited to the command and data handling application. Therefore, we illustrate the implementation of the same in next section.

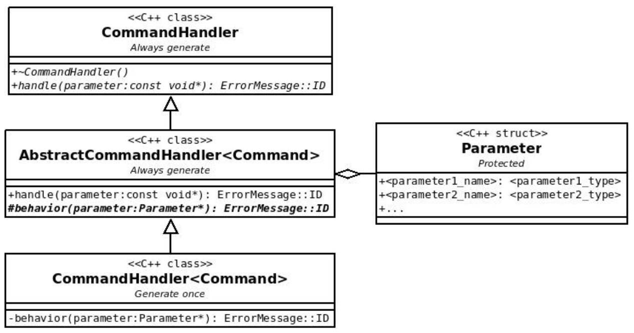

<<C++ class>>
CommandHandler
Always generate
+~CommandHandler()
+handle(parameter:const void*): ErrorMessage::ID
<<C++ class>>
AbstractCommandHandler<Command>
Always generate
+handle(parameter:const void*): ErrorMessage::ID
#behavior(parameter:Parameter*): ErrorMessage::ID
<<C++ struct>>
Parameter
Protected
+<parameter1_name>: <parameter1_type>
+<parameter2_name>: <parameter2_type>
+...
<<C++ class>>
CommandHandler<Command>
Generate once
-behavior(parameter:Parameter*): ErrorMessage::ID

Figure 5. GGP based Command Handler

The command and data handling application software is organized hierarchically. Commands are grouped into components, and components are grouped into applications as shown in Figure 4.

Most of the target code is automatically generated from the model created using DSLs (see Section 4). However, for some of the software components, generated and manually written code must be integrated. The framework follows the Generation Gap Pattern (GGP) [17] to separate automatically generated and manual code. The idea is to put purely generated files in a folder named *src-gen* and to have files which are generated only once (e.g. class stubs, subsystem specific detailed software implementation) in a different folder named *src* that allows the user to modify them. Figure 5 shows how the automated (*Always generate*) and manual (*Generate once*) code generation is integrated together based on GGP.

The implementation details of language grammar and code generation are illustrated in the next section.

4. Implementation

The grammar for the set of domain specific languages described in this paper is developed by using the widespread open source framework Eclipse and Xtext [18]. A DSL is described in Xtext by using an EBNF (Extended Backus-Nauer Form) style grammar. From this, a parser for the DSL with an internal representation and a code editor is generated. The generated DSL editor supports syntax highlighting, code folding, content assist, and inline error markers. Additional checks such model validation, content assist feature and code formatting capabilities is added by providing simple Java extensions to the Xtext project.

This section provides an overview of implementation. The complete implementation details are available in [19].

Grammar

As a first step, a grammar for common DSL is defined as shown in the Listing 1 .

It covers number literals, qualified identifiers and documentation comments which form the basis for the application specific DSLs. Each terminal is in turn then defined. It provides abstraction on top of default terminals provided by Xtext shown in Listing 2.

Based on the common grammar and the semantic model shown in Figure 4, higher level elements of the application data model are defined. They contain a collection of the

Listing 1. Definition of Common Grammar

```
/* Common terminals and data types
 * for all AOCS grammars
 */
generate common
"http://www.dlr.de/lambda/dsl/common/Common"

import
"http://www.eclipse.org/emf/2002/Ecore" as
     ecore

/* Production rule
 * Needed to generate EPackage
 * and for testing
 */

CommonModel: 'common' name=ID
(
    'IntValue' intValue=IntValue
   | 'SignedInt' signedInt=SignedInt
   | 'RealValue' realValue=RealValue
   | 'HEX' hex=HEX
   | 'QualifiedID' qid=QualifiedID
);
```

Listing 2. Definition of Data Types

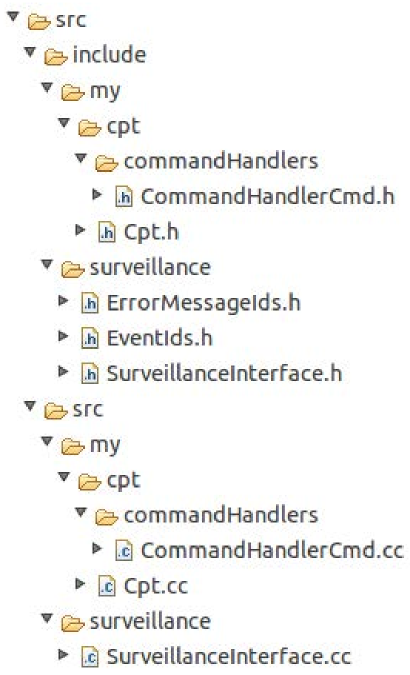
```
/* Data type rules */

QualifiedID
returns ecore::EString: ID ('.' ID)*;

IntValue
returns ecore::EInt: SignedInt | HEX;

/* Data types with value converters
 */

SignedInt returns
ecore::EInt: ('+'|'-')? INT;

RealValue
returns ecore::EDouble: SignedInt '.' INT
     EXPONENT?;
```

elements which are immediately below them in the hierarchy. By taking the advantage of the cross-referencing mechanism of Xtext, we get some extra freedom while defining the hierarchy. The grammar is listed as follows in Listing 3.

Using an application level grammar, a mission specific subsystem level grammar is defined. For example, the AOCS subsystem grammar defining data model for command and data handling application is shown in Listing 4.

Defining these grammar rules is a one time task. We have tried to include all the possible rules based on the previous experience. We have tried to keep it as simple and basic as possible by splitting the complex data structure to the atomic unit level. Of course the rules can be modified or extended as per the special requirements of the individual mission.

After compiling grammar rules, our domain specific language is ready to use for defining the actual mission data model. Eclipse provides DSL specific editors where individual domain expert can define the data model corresponding to the domain. As an example, Listing 5 shows the definition of a AOCS data model based on the AOCS grammar defined in Listing 4.



```
src
  include
    my
      cpt
        commandHandlers
          CommandHandlerCmd.h
        Cpt.h
      surveillance
        ErrorMessageIds.h
        EventIds.h
        SurveillanceInterface.h
  src
    my
      cpt
        commandHandlers
          CommandHandlerCmd.cc
        Cpt.cc
      surveillance
        SurveillanceInterface.cc
```

Figure 6. Generated Customizable files

Model Verification

A big advantage of model-driven development of software is the practical verification on model level. To support the requirements for safe and reliable software the model is checked against several constraints. This constrains defined by the DSL enable better type safety and range checks to achieve more reliable code.

Listing 4. AOCS Grammar

```
grammar de.dlr.lambda.dsl.aocs.AOCS with
de.dlr.lambda.dsl.common.Common
hidden(WS, ML_COMMENT, SL_COMMENT)

generate aOCS
       "http://www.dlr.de/lambda/dsl/aocs/AOCS"
import "http://www.eclipse.org/emf/2002/Ecore"
       as ecore

AocsModel:
    (applicationDefs+=Application
       | componentDefs+=Component
       | commandDefs+=Command
       | parameterDefs+=Parameter
       | enumerationDefs+=Enumeration
       | rangeDefs+=Range
    )*
;
```

Code Generation

From the developers of Xtext, the Xtend [20] framework is developed to generate source code files from the model created using Xtext grammar definition. Xtend is a statically typed programming language and has strong template features to create code generators. Therefore the same is used in this case for generating C++ source code. The Xtend template maps the grammar rules into C++ syntax rules along with semantic implementation. Based on the model file created in Listing 4 and concept described in Figure **??**, the C++ project structure with required folders and source files is generated

Listing 3. Definition of Application Grammar

```
Application:
    comment=DOC_COMMENT?
    'application' name=ID 'is'
        (enumerationDefs+=Enumeration
            | rangeDefs+=Range
            | parameterDefs+=Parameter
            | commandDefs+=Command
            | componentDefs+=Component
            | 'component' componentRefs+=[Component|QualifiedID]
        )*
    'end' 'application'
;

Component:
    comment=DOC_COMMENT?
    'component' name=QualifiedID 'is'
        (enumerationDefs+=Enumeration
            | rangeDefs+=Range
            | parameterDefs+=Parameter
            | commandDefs+=Command
            | 'command' commandRefs+=[Command]
        )*
    'end' 'component'
;

Command:
    comment=DOC_COMMENT?
    'command' name=ID 'is'
        (enumerationDefs+=Enumeration
            | rangeDefs+=Range
            | parameterDefs+=Parameter
            | 'parameter' parameterRefs+=[Parameter]
        )*
    'end' 'command'
;

Parameter:
    comment=DOC_COMMENT?
    'parameter' name=ID 'is'
    type=Type (array?='array' '(' arraySize=INT ')')?
    (
        (constrained?='in' (
            rangeConstrained?='range' (range=AnonymousRange | rangeRef=[Range])
            | enumConstrained?='enum' (enumeration=AnonymousEnumeration | enumerationRef=[Enumeration])
        ))?
        & ('with' 'units' unit=Unit)?
    )
;
```

automatically.

As described in Section 3, the code generation follows so called *Generation Gap Pattern*. According to it, two types of files are generated such as:

• *src* : This folder receives all the files which follow the generate once policy. They are not overwritten or deleted, and the user may edit them as needed. It mainly consists of command handlers, the base class and specific classes (see Figure 6).
• *src-gen* : Purely generated files are written to this folder. All the resources in it are cleaned and re-written every time the code generator runs. Additionally, the files are marked as 'derived' so that the user is warned by Eclipse when trying to edit them (see Figure 7).

5. Conclusion and Future Work

The DSL based framework presented in this paper provides a stable base to start developing the CDH. The set of data description language is designed to cover common functionalities to all satellite subsystems and a simple hierarchical semantic model for CDH functionality. By using a data description DSL, a unified data model for CDH is created and required code is automatically generated from it. Because of the cross-referencing capabilities of Xtext, domain experts get flexibility to separate a model definition by levels of abstraction. The 'generate once' pattern was chosen to provide the separation of manual and automatic code.

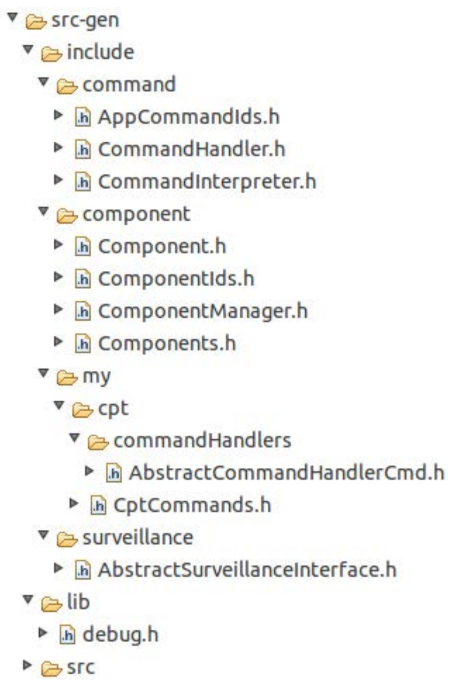
Frequently used functionalities like data serialization and protocol wrapping are encapsulated into a library which is then utilized from the generated code. In this way, knowledge from previous missions is maintained and can be reused in future missions.

Such DSL-based models of data structures of a spacecraft's on-board system reduces design and programming efforts. It also provides verification of models which results into less error-prone and high quality software.

The DSL-based approach provides a standard process to define a general data structure. But at the same time, it gives enough flexibility to tailor it to the mission specific requirements.

Listing 5. TET-1 Mission specific AOCS Data Model

```
/* Example AOCS model */

/* Documentation comments preceding applications, components,
 * commands or parameters will be included in the target code
 */
application app is

   /* Components can be given qualified names for the purpose
    * of organizing the generated files, but they will be treated
    * by their last name only, so it must be unique.
    */
   component my.cpt is

      // Command definition
      command cmd is

         // Parameter definition
         parameter par1 is float in range 0.0 to 1.0 with units ms

         /*
          * Components, commands, parameters, ranges and enums can also
          * be referenced, as long as their definition is in scope
          */
         parameter par2
         parameter par3 is int32 in enum ENUM

      end command

   end component

end application

// Definition of parameter to be referenced within a command
parameter par2 is bool

// Definition of enum to be referenced within a parameter
enum ENUM is (ZERO, TWO=2, THREE)
```



```
▼ src-gen
  ▼ include
    ▼ command
      ▶ AppCommandIds.h
      ▶ CommandHandler.h
      ▶ CommandInterpreter.h
    ▼ component
      ▶ Component.h
      ▶ ComponentIds.h
      ▶ ComponentManager.h
      ▶ Components.h
    ▼ my
      ▼ cpt
        ▼ commandHandlers
          ▶ AbstractCommandHandlerCmd.h
        ▶ CptCommands.h
    ▼ surveillance
      ▶ AbstractSurveillanceInterface.h
  ▼ lib
    ▶ debug.h
  ▶ src
```

Figure 7. Purely Generated files

As mentioned in the Section 2, the Space Avionics Open Architecture (SAVOIR) is an initiative for standardization of the system architecture and component interfaces. It supports DSL-based development. Therefore our solution fits well with the ongoing efforts in the same direction. In future, we would like to make our data description language compatible with SAVOIR.

The integrated design environment named *Virtual Satellite* (VirSat) [21] is designed to support the design process of spacecraft systems in a concurrent engineering way starting from Phase 0/A. Based on the model based systems engineering and the object oriented approach, VirSat creates and maintains the consistent system model and subsystem dependencies. In the case that the data description model shows promising results, its development may be continued with the final goal of being integrated in the VirSat framework and linked to the common system model throughout the different phases of spacecraft development.

In future, automatic test case generation and code verification could be added to reduce testing efforts. The data model can also form a basis for implementing the ground support software in future. While keeping the data model extensible, new functionalities can be added in future. That makes it usable for many applications.

ACKNOWLEDGMENTS

We thank Olaf Mailbaum, Senior Scientist from the Department of Simulation and Software Technology at DLR, Braunschweig, for providing files from TET-1 Satellite mission for

analyzing the AOCS, CDH and data description files.

REFERENCES

[1] S. Foeckersperger, K. Lattner, C. Kaiser, S. Eckert, S. Ritzmann, R. Axmann, and M. Turk, "The on-orbit verification mission tet-1 – project status of the small satellite mission & outlook for the one year mission operation phase," in *4S Symposium (Small Satellites Systems and Services)*, Funchal, Portugal, May-June 2010.

[2] *Ground systems an operations - Telemetry and telecommand packet utilization*, European Cooperation for Space Standardization Std. ECSS-E-70-41A, January 2003. [Online]. Available: http://www.ecss.nl/forums/ecss/dispatch.cgi/standards/showFile/100303/d20040512075344/No/ECSS-E-70-41A%2830Jan2003%29.pdf

[3] *TC Space Data Link Protocol*, Consultative Committee for Space Data Systems Std., September 2003. [Online]. Available: http://www.ccsds.org/

[4] O. Maibaum, T. Terzibaschian, C. Raschke, and A. Gerndt, "Software reuse of the bird acs for the tet satellite bus," in *8th International Symposium of the International Academy of Astronautics (IAA)*, R. Sandau, H.-P. Röeser, and A. Valenzuela, Eds., vol. 8. Berlin: Wissenschaft und Technik Verlag, April 2011.

[5] P. H. J. Feiler, "Toward model-based embedded system validation through virtual integration," *Software Tech News*, vol. 12(4), pp. 26–32, January 2012, the Data & Analysis Center for Software, United States Department of Defense.

[6] N. Basha, S. Moiz, and R. M., "Model based software development: Issues & challenges," *Special Issue of International Journal of Computer Science & Informatics (IJCSI)*, vol. 2, pp. 226–230, 2012.

[7] R. Viennau, "Model-driven development: Better systems through advanced automation," *Software Tech News*, vol. 12(4), January 2010, the Data & Analysis Center for Software, United States Department of Defense.

[8] M. Voelter, *DSL Engineering - Designing, Implementing and Using Domain-Specific Languages.* CreateSpace Independent Publishing Platform, January 2013, no. ISBN 978-1-4812-1858-0. [Online]. Available: www.dslbook.org

[9] S. Gretlein. (2012, October) Modeling of embedded designs - part 1: Why model? National Instruments. [Online]. Available: http://www.embedded.com/design/prototyping-and-development/4399743/Modeling-of-embedded-designs--Why-model--

[10] T. J-P., "Savoir: Reusing specifications to improve the way we deliver avionics," in *Embedded Real Time Software and Systems*, 2012. [Online]. Available: http://www.erts2012.org/site/0p2ruc89/6c-1.pdf

[11] A. Jung and J.-L. Terraillon. (2010, December) Faster, later, softer: Cordet – an on-board software reference architecture. Workshop on Spacecraft Flight Software (FSW-10). European Space Agency ESTEC, Software Systems Division. [Online]. Available: http://flightsoftware.jhuapl.edu/files/2010/FSW10_Jung.pdf

[12] L. J. Hansen, P. Graven, D. Fogle, and J. Lyke, "The feasibility of applying plug-and-play concepts to spacecraft guidance, navigation, and control systems to meet the challenges of future responsive missions." in *7th International ESA Conference on Guidance, Navigation & Control Systems*. Los Angels, USA: American Institute of Aeronautics and Astronautics (AIAA), April 2008.

[13] O. Dubuisson, *ASN.1 — Communication Between Heterogeneous Systems*. Morgan Kaufmann, 2000.

[14] K. Doerfler and M. Voelter. (2011, November) Programming refrigerators with eclipse xtext.

[15] M. Koets and M. Lecocke, "A specialized programming language for coordinating software execution timing in embedded systems," in *Aerospace Conference, 2014 IEEE*, March 2014, pp. 1–15.

[16] S. Venigalla, B. Eames, and A. McInnes, "A domain specific design tool for spacecraft system behavior," in *The 8th OOPSLA Workshop on Domain-Specific Modeling (DSM'08)*, 2008.

[17] H. Behrens. (2009, April) Generation gap pattern. Accessed on 23.10.2014. [Online]. Available: http://heikobehrens.net/2009/04/23/generation-gap-pattern/

[18] *Xtext Documentation*, Itemis, September 2014. [Online]. Available: http://www.eclipse.org/Xtext/documentation/2.7.0/XtextDocumentation.pdf

[19] P. Isidro, "Automatic code generation for attitude and orbit control systems using domain-specific languages," December 2014, Master Thesis.

[20] S. Efftinge and S. Zarnekow, "Extending java – xtend: a new language for java developers," *PragPub, The Pragmatic Bookshelf*, no. 30, pp. 5–11, December 2011.

[21] M. Deshmukh, V. Schaus, P.-M. Fischer, D. Quantius, V. Maiwald, and A. Gerndt, "Decision support tool for concurrent engineering in space mission design," in *Concurrent Engineering Approaches for Sustainable Product Development in a Multi-Disciplinary Environment*, vol. 1, no. 5, ISPE International Conference on Concurrent Engineering 2012. Trier, Germany: Springer - Verlag London, September 2012, pp. 497–508.

BIOGRAPHY

Meenakshi Deshmukh *works since June 2011 for the German Aerospace Center (DLR) within the Department of Simulation and Software Technology. She joined the Software for Space Systems section. Since then she is working on the software development projects for supporting the Early Design and Analysis of Space Missions. Her scientific research focuses on the model-based system and software engineering. She has completed her bachelor studies in Information Technology from Pune, India and then pursued Master's degree in Computer engineering from University of Duisburg-Essen, Germany.*

Benjamin Weps *received his M.Sc in Computer Science from the University of Applied Sciences Bingen in 2013. Afterwards he started as a research assistant at the German Aerospace Center (DLR). At the department "Software for Space Systems and Interactive Visualization", he is working on software for space systems. His current research interests focuses on embedded systems for space applications.*

Pedro Isidro *is an Aerospace Engineering student at Instituto Superior Técnico Lisbon, Portugal. He is currently working on his master's thesis at the department of Simulation and Software Technology of the German Aerospace Center (DLR). The goal of the thesis is the development of a Domain-Specific Language (DSL) for the Attitude and Orbit Control System (AOCS) of spacecraft. The product is a textual programming language optimized for the AOCS, associated with editor support and a C++ code generator.*

Andreas Gerndt *is the head of the Department Software for Space Systems and Interactive Visualization at the German Aerospace Center (DLR). He received his degree in computer science from Technical University, Darmstadt, Germany in 1993. In the position of a research scientist, he also worked at the Fraunhofer Institute for Computer Graphics (IGD) in Germany. Thereafter, he was a software engineer for different companies with focus on Software Engineering and Computer Graphics. In 1999 he continued his studies in Virtual Reality and Scientific Visualization at RWTH Aachen University, Germany, where he received his doctoral degree in computer science. After two years of interdisciplinary research activities as a post doctoral fellow at the University of Louisiana, Lafayette, USA, he returned to Germany in 2008.*