Formal Modeling of Data. A Case Study for Space Applications[1]

Jean-Paul Blanquart[1], Gérard Bulsa[2], David Lesens[3], George Mamais[4], Maxime Perrotin[2]

[1] Astrium Satellites. 31, avenue des cosmonautes, F-31 402 Toulouse Cedex 4, France
Jean-Paul.Blanquart@astrium.eads.net
[2] ESTEC. Keplerlaan 1. PO Box 299, NL-2200 AG Noordwijk, The Netherlands
{Gerard.Bulsa, Maxime.Perrotin}@esa.int
[3] Astrium Space Transportation. Route de Verneuil, BP 3002, F-78 133 Les Mureaux Cedex, France
David.Lesens@astrium.eads.net
[4] Semantix Information Technologies, K.Tsaldari 62, Poligono 114 76, Athens, Greece
gmamais@semantix.gr

Abstract. This paper reports on a case study investigating the interest of formal data modeling approaches for on-board space software. The objectives are to complement formal approaches on functional parts and solve difficulties coming from lack of formalism in the description and handling of data exchanged between different parts of software. The first part addresses the analysis of space software data and their classification into families (section 2) and the elaboration of an ASN.1 data model for a case study representative of these data families (section 3). The second part illustrates two important outcomes of such formal data models: the capability to automatically generate "Interface Control Documents", the contractual documents describing software interfaces and data (section 4) and automatically generate the needed interfacing code with the appropriate format encoders and decoders (section 5).

1 Introduction

The use of modeling techniques for describing on-board and ground space systems has been identified by many studies as an appropriate way to master the complexity of software. Today, projects show an increasing interest for applying a model-based approach, from which several benefits are expected:

- A clear and unambiguous representation of the software architecture;
- A complete and verifiable representation of the software behavior;
- A reliable implementation of the software.

There exist several languages and tools that have been assessed as good candidates to fulfill most of these needs. These languages allow the description of a system's logical and physical architecture as well as its behavior, and tools provide simulation, model checking, automatic test and code generation.

However, a system is generally not developed in an independent manner by a single developer team. Whether they use modeling techniques or not, many projects encounter

[1] This work has been partially funded by the ASSERT project (Contract n°: 004033 of FP6-2003-IST-2) [ASSERT] and the ESA project SSCDMT "System Software Co-engineering: Data Modeling Technologies" (Contract ESTEC 20467/06/NL/JD) [SSCDMT].

many difficulties integrating different communicating pieces of software or different sub-systems, especially concerning exchanges of data:

- Two communicating applications may exchange a piece of data, without agreement on the data semantics. For example Application 1 defines a structure of data in one way and Application 2 in another way which don't fit together. A similar and frequent case is when different set of values for an enumerated data type are given in different parts of the software.
- The application semantics may be consistent but the way the data is physically encoded is incompatible. This is the typical problem of little/big endian in an heterogeneous environment, but it can also be that one side always encodes an integer in 32 bits and that another side would try to use a more compact and context-dependent binary encoding.

The first problem essentially comes from the fact that the importance of data typing is sometimes overlooked, especially when using coding languages like C which do not impose themselves strong constraints on type consistency. Other languages such as Ada, widely used in space transportation, provide some solutions, though they do not address all the issues (e.g., they do not define and enforce encoding rules) at all needed levels (they are implementation languages, not modeling). Additionally, Ada is unfortunately not accepted by numerous companies and less and less tools are available for this language. Moreover, "popular" modeling languages like UML or SIMULINK rarely generate Ada code and do not include precise semantics nor syntax for what concerns data typing.

Defining precise application semantics for data and encoding rules in a standardized, abstract manner that would be independent from a low-level implementation language is a concern that is rarely taken into account and current space standards hardly mention this need.

Yet, languages and tools exist to ensure global data consistency even in heterogeneous environments. The Abstract Syntax Notation One (ASN.1) is the most widely used data modeling language in this family [ASN1]. It is today used on some major ESA projects but mainly for the ground segment part. Other similar languages exist but their tool support is generally weak or they lack essential features (such as binary encoding in the case of XSD or easy integration with automatically generated functional code in the case of IDL). ASN.1 therefore appeared as the best candidate to be investigates for onboard, real-time software. It is an internationally recognized standard that is supported by a wide variety of tools, including open-source.

The objectives of this paper are to report on investigation, prototyping and experimentation on data modeling methods, techniques and tools, as a possible solution to improve the way data is handled in the whole lifecycle of a space project.

The paper is split in two main parts. The first part (sections 2 and 3) collects and analyses data, data families and related difficulties, from real space projects, and reports on the elaboration of formal data models in ASN.1 on a selected case study representative of space development. The second part (sections 4 and 5) is dedicated to the development of a tool to automatically generate Interface Control Documents from a formal data model, and to the integration of data modeling methods and tools with other modeling methods, tools and automatic code generators used in software development.

2 Identification of data sharing (families of data)

2.1 Introduction

The objective of this section is to perform an analysis of user needs by sorting all the data of the software development process which can be exchanged:

- Between the software and its external environment;
- Between components inside the software;
- Between different teams involved in the software development;

An interface can be characterized by the following two aspects:

- The semantic of the data structure to exchange; we can consider this aspect as a functional description and organization of the data structures;
- The encoding of the data structure, which can be considered as the implementation aspect.

2.2 Families of data

The analysis performed in the scope of the study presented in this paper has defined the following families of data:

1. Interface between two teams. The first team defines the need, and thus has to describe data in an abstract manner (without useless constraints on the implementation). The second team defines the implementation, in accordance with the software design.
2. Interface between two programming languages or between two pieces of code automatically generated from two modeling tools. Each programming language and each modeling tool has its own capabilities and constraints to describe interfaces. The data modeling language shall be compatible with the programming languages features and the modeling tools. The modeling language shall be supported by a tool able to automatically:
    - Generate the interfaces in the two programming and modeling languages.
    - Generate the glue code allowing a message passing between the two programming languages or the code automatically generated from modeling languages.
    - Verify that the interfaces in the two programming or modeling languages are still consistent after a modification.

   For programming languages, generating and verifying interfaces are generally easily achieved by generating a specification file (".h" in C or ".ads" in Ada). For modeling tools, it is more complicated. The tool shall:
    - Be able to generate the interface description in the proprietary modeling language format.
    - Be able to read and understand the proprietary modeling language format in order to verify the consistency of interfaces.

   The most common languages to be supported are: C, Ada, Scade, Simulink, Rhapsody

3. Interface with high bandwidth, data range checks and no dynamic parameterization (for instance interfaces equipment / processor). This kind of interface shall be optimized for CPU limitation. It can be hard coded.
4. Interface with high bandwidth, no data range checks and dynamic parameterization (for instance telemetry). This kind of interface shall be optimized for CPU limitation. It shall be table driven in order to allow dynamic parameterization during the flight.
5. Specific sub-cases of the previous two cases (interface processor / equipment). The functional constraints are identical, but the bandwidth is more limited. The same implementation can be chosen, even if there are less CPU constraints.
6. Interface without encoding (between two functional blocks in the same software in the same programming language). In order to save CPU, no specific encoding shall be used (or more precisely, the same encoding shall be use by the two communicating parties).
7. Interface with a specific encoding (interface imposed by one of the two communicating entities). A standard encoding cannot be used because the specific encoding is imposed.

The following figure summarizes the defined families of data.

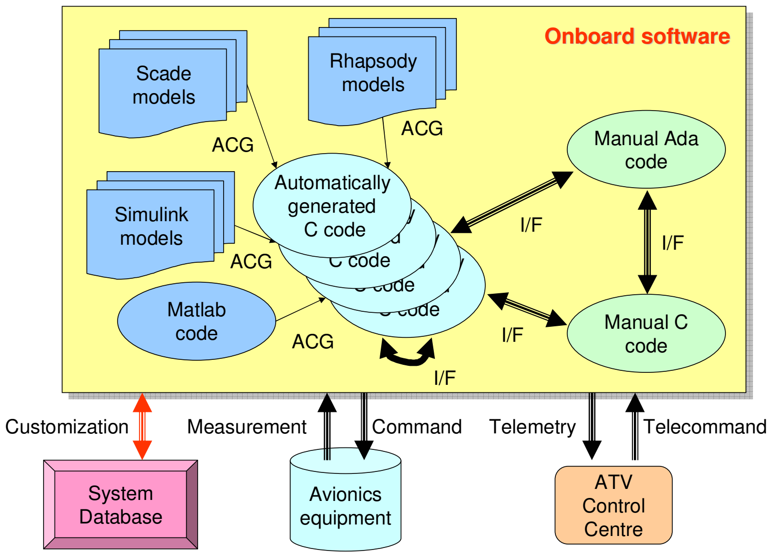


FIG. 1 – *Synthesis of data families.*

3 Data modeling with ASN1 in space domain, case studies

3.1 Introduction

This section reports on the development of a case study covering the different features listed in section 2. It focuses mainly on the following two points:

- The PUS (Packet Utilization Standard) [PUS].
- The interface between different modeling tools.

3.2 Interfaces between the spacecraft and the ground control centre – the Packet Utilization Document

The external interfaces are composed of the interface ground control centre / board, i.e. telecommand (TC) and of the interface board / ground control centre, i.e. telemetry (TM). In the European Space Industry, the format of telecommand and telemetry are standardized in the Packet Utilization Standard (PUS). In the scope of this paper, we have analyzed a specific version of the PUS developed for the ATV (Automated Transfer Vehicle) project, the Packet Utilization Document (PUD).

3.2.1 Parameters depending on a previous field value

Some field definitions of a PUD packet can depend of the value taken by a previous field. For instance (PUD (1, 2) service (TC Acceptance Report – Failure)):

Packet Identification	**Packet Sequence Control**	**Code**	**Parameters**
16 bits	**16 bits**	**Enumerated 32 bits**	**Any**
			Depending on the Code field

TAB. 1 – *PUD service (1, 2)*

In ASN.1, the couple (error code ID / parameters) can be replaced by a single CHOICE structure. Compared to the classical "union" of the C language, the CHOICE structure add in its semantics the encoding of the selected field (the "code" field of Tab.1 becomes useless). The PUD format is then not any more strictly respected but the data representation becomes lighter and more formal.

```
RESPONSE-PARAMETER ::= CHOICE {
        packet-accepted-id NULL,             -- The packet is accepted by the FAS
        far-error-id SEQUENCE {              -- Error due to the FAR (at CPF level)
                far-cpf1 FAR,
                far-cpf2 FAR
        },
        bad-apid APID,                       -- Unknown Extended APID
```

TAB. 2 – *Modeling of PUD service (1, 2)*

3.2.2 List of parameters

The definitions of some fields can depend completely on the value of a previous field, for instance in the PUD service (2, 2) "Distribute Low-Level Commands".

LLC ID	**LLC Data**
Enumerated	**Any**

TAB. 3 – *PUD service (2, 2)*

As before, the ASN.1 language allows modeling at the same time all the possible values of the LLC data. The LLC ID can then be suppressed.

```
tc-2-2 SEQUENCE {
        llc CHOICE {
                tmtc-current-state ENUMERATED { tmtc-m-surv-free-flight,
                        tmtc-m-surv-attached, tmtc-m-free-flight }
                tmtc-enable-dump-flag BOOLEAN
                tmtc-abort-dump-memory NULL
                tmtc-al-prox-loss-threshold INT32
        }
}
```

TAB. 4 – *Modeling of a PUD service (2, 2)*

3.2.3 Array of variable size

The last example coming from the PUD is an array with a variable size. In the PUD service (14, 14) “Disable Transmission of Event Reports”, the RID parameter is an array of size N, which is an other parameter of the packet.

N	RID
Unsigned Integer	**Enumerated**
	<---- Repeated N times ---->

TAB. 5 – *PUD service (14, 14)*

In ASN1, the “N” parameter becomes useless, since it is included in the encoding of the packet.

```
-- Disable Transmission of Event Reports
tc-14-14 SEQUENCE {
        -- The Report Identifier identifying an event report packet definition
        rid SEQUENCE SIZE(1..24) OF RID
},
```

TAB. 6 – *Modeling of a PUD service (14, 14)*

3.3 Interface internal to the software, with different modeling tools

Space onboard software development generally follows a Model Driven Approach (MDA) because this allows an early validation of each stage of development.

Moreover, some kinds of models and tools allow the automatic code generation from the model. In this case, the classical V development cycle becomes a Y development cycle and the development costs are dramatically decreased, especially the costs of taking evolutions into account. However, it raises specific problem for data and interface description.

3.3.1 Application of MDA to a space onboard software development

Two parts in the software have to be distinguished:

- Application software behavior (mission management, Fault Detection Isolation and Recovery, TMTC…). This can be represented with state machines (e.g. SDL, UML);

- Numerical algorithms: Guidance Navigation Control, Thermal Control…
    - Static architecture: hierarchical decomposition, interface (input/output of each block), data-flow between blocks: Simulink or Scade;
    - Dynamic architecture: Period and condition of activation, CPU consumption: Scade;
    - Numerical equations: C, C++, Ada or embedded-MATLAB.

The problem of interfacing two different modeling languages is raised at two levels: Modeling level and code level.

In the case study reported in this paper, a Scade model describes the architecture of the software and the event driven part of the software (mission management). The algorithmic leaves of this architecture are modeled in Simulink.

3.3.2 Scade / Simulink communication

The main difference between the codes generated by RTW-EC (Real Time Workshop Embedded Coder) from Simulink models and by KCG (Qualified Code Generator) from Scade models is the implementation of data types, especially of array data types. One characteristic of the code generated by Scade (up to version 5) is that the arrays and the matrixes are implemented as structure in the following manner. The following table gives an example of a matrix (3, 2).

```
let type Types
   T_INIT_EFF_MATRIX = real ^3 ^2;
tel ;
```

TAB. 7 – *Example of matrix in Scade*

The corresponding C codes in respectively Scade version 5.1 and 6 are the following:

typedef struct {
   real _F0;
   real _F1;
   real _F2;
} _T15_B00_MSU;
typedef struct {
   _T15_B00_MSU _F0;
   _T15_B00_MSU _F1;
} _T20_B00_MSU;
typedef _T20_B00_MSU T_INIT_EFF_MATRIX;
	typedef _real array_37[3];
	typedef array_37 array_47[2];
	array_47 T_INIT_EFF_MATRIX;

TAB. 8 – *C code generated for matrix in Scade 5.1 and 6*

The same matrix data type will be implemented by RTW EC as a linear vector.

```
real_T FORCE_EFFICIENCY_MATRIX[12];
```

TAB. 9 – *C code generated for matrix by RTW-EC*

Running the Scade code with the Simulink code will require manually coding a wrapper between the two generated codes.

```
S_COMPUTE_VELOCITY0_U.EFF_MAT[ 0 ] = EFF_MAT->_F0._F0;
S_COMPUTE_VELOCITY0_U.EFF_MAT[ 1 ] = EFF_MAT->_F0._F1;
S_COMPUTE_VELOCITY0_U.EFF_MAT[ 2 ] = EFF_MAT->_F0._F2;
S_COMPUTE_VELOCITY0_U.EFF_MAT[ 3 ] = EFF_MAT->_F1._F0;
S_COMPUTE_VELOCITY0_U.EFF_MAT[ 4 ] = EFF_MAT->_F1._F1;
S_COMPUTE_VELOCITY0_U.EFF_MAT[ 5 ] = EFF_MAT->_F1._F2;
```

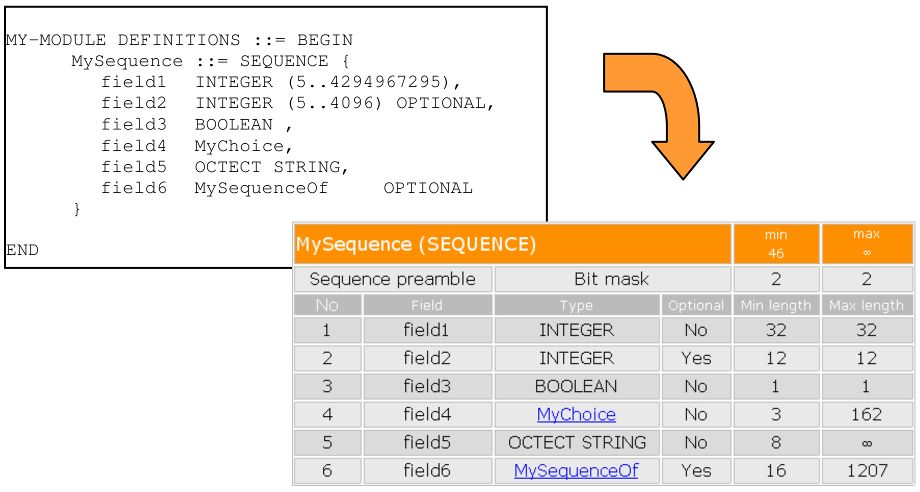
TAB. 11 – *Wrapper between Scade and Simulink*

Modeling the Simulink/Scade interface in ASN1 is trivial. However, this modeling allows the automatic generation of consistent ICD (section 4) and wrapper (section 5).

4 Automatic Interface Control Document Generator

An Interface Control Document (ICD) is a document identifying interface data that is exchanged between software components. ICDs typically allow a visual overview of the data structures and the frames that are exchanged between applications. Such documents are authored manually after a cumbersome inspection of the data structures themselves at the code level and after also accounting for the marshalling logic that will transform them to the wire format. Therefore, the process of creating an ICD is error-prone and update of these documents is awkward especially in projects with big data models that often change.

The automatic ICD generator is command line tool that takes as input a formal data model in ASN.1 and produces the equivalent ICD automatically. The generated document contains tables for each type defined in the ASN.1 data model and each table has as many rows as the number of the ASN.1 fields (see Figure 2). The minimum and maximum numbers for each field represent the min and max number of bits required to encode the specific field using the Packed Encoding Rules (PER).



```
MY-MODULE DEFINITIONS ::= BEGIN
      MySequence ::= SEQUENCE {
         field1   INTEGER (5..4294967295),
         field2   INTEGER (5..4096) OPTIONAL,
         field3   BOOLEAN ,
         field4   MyChoice,
         field5   OCTECT STRING,
         field6   MySequenceOf      OPTIONAL
      }
END
```

MySequence (SEQUENCE)				min 46	max ∞
Sequence preamble		Bit mask		2	2
No	Field	Type	Optional	Min length	Max length
1	field1	INTEGER	No	32	32
2	field2	INTEGER	Yes	12	12
3	field3	BOOLEAN	No	1	1
4	field4	MyChoice	No	3	162
5	field5	OCTECT STRING	No	8	∞
6	field6	MySequenceOf	Yes	16	1207

FIG. 2 – *Example input and output of automatic ICD generator*

The format of the generated document can be HTML, RTF and LaTeX. Figure 3 depicts the operational model of the Automatic ICD Generator (depicted as a black box in this figure).

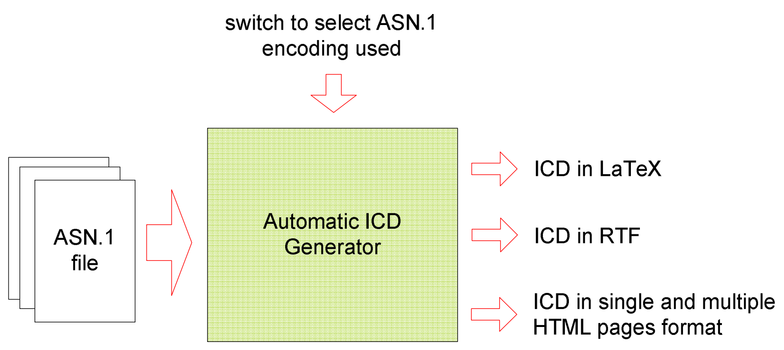

switch to select ASN.1 encoding used
ASN.1 file
Automatic ICD Generator
ICD in LaTeX
ICD in RTF
ICD in single and multiple HTML pages format

FIG. 3 – *Operational model of the ICD automatic generator*

As can be seen in Figure 3 the Automatic ICD Generator accepts as input:

- The ASN.1 grammar describing the data to be exchanged (note that the ASN.1 grammar may be provided in more than one ASN.1 files linked together with ASN.1 "import" statements).
- A set of command line parameters that control the behavior of the ICD generator like the format of the generated document etc.

5 Automatic Code Generator

5.1 Introduction

In classical approaches, with existing tools, it is only possible to generate a complete application if it is modeled in one single tool and if it runs on a single processor, preferably on a commercial operating system. In that particular case, code generators from modeling tools can generate a set of tasks in a proprietary manner that can run on a predefined target.

In practice however, this case never happens. Applications are developed by several industries, using different, specialized languages or modeling tools (e.g. Matlab for Attitude and Orbital Control System algorithms, UML or SDL for state machines, manual code), and obviously these pieces of code have to exchange data. Moreover, operating systems that are supported by the code generators are never the ones that are used in real projects (for space systems at least). This results in a huge manual intervention in the process in order to glue everything together, and to create tasks that run properly on the chosen operating system. If several processors are used (case of a distributed system in a heterogeneous environment) this complicates even more the situation because extreme care has to be put on the representation of data (need to handle little/big endian, etc).

The ASSERT and SSCDMT projects propose solutions to automate this complete process. This section reports how this system was concretely implemented and tested on a case study, using two different and complementary modeling tools for producing the functional code.

5.2 Statement of the problem

In a space system it is very common to use synchronous, data-flow oriented languages such as Scade or Matlab/Simulink to define control laws and mathematical algorithms. These languages provide all the constructs for this purpose, together with good simulation and verification facilities.

On the other hand, to specify the behavior of the system (orchestration of the calls, decisions based on results from the algorithms, fault detection and recovery, protocols, etc.), state-machines are more appropriate. Languages such as SDL and tools such as Rhapsody or Statemate are sometimes more suitable to capture the behavior of the system than Scade or Matlab for example (even if the SSM or Stateflow can be an alternative). It is therefore logic and common to use the most adapted language for each specific purpose, since there is no single, universal language that can handle all facets of a system.

Having this in mind, we need to:

1. **Have, at system level, a single representation of all the data types** (otherwise, how to be sure that both implementers will share the same semantics for the data?)
2. Decide, at system level, of **the representation of all the data types when it is carried between different blocks**. We need to have a neutral binary representation of the data independent from the physical architecture on which each node resides.
3. Have a way to **go from one representation of a data (Scade, SDL or Simulink… at code level) to another one representation of the same data**.
4. **Have an operating system with standardized interfaces** so that it is possible to automate the connection between code generated by a functional modeling tool and the operating system itself.

The ASSERT and SSCDMT projects propose a process and a tool chain solving this issue. The main tool of the tool chain has been developed by Semantix Technologies, but the developed technology is generic and can be adapted to any modeling tool and code generator (for instance, an adaptation to ObjectGeode has been developed by ESA, an adaptation to Scade has been developed by Esterel Technologies).

The following sections show how these issues have been solved.

5.3 Representation of the data types at system level

The data types shared at system level have been modeled in ASN.1 (see section 3.). Thus, the tool chain developed by the Semantix Information Technologies Company provides means to convert ASN.1 to **any** particular modeling language, **preserving the semantic of the data**.

For ObjectGeode the conversion was straightforward (ObjectGeode natively supports ASN.1 with little restrictions). For SCADE some more important modifications have automatically been performed to conform to the Lustre data type constraints.

5.4 Platform-independent representation of the data and data transformation

This essential point can be very difficult to address in theory because it implies to have data encoders and decoders (also called marshallers or serializers) accessible from each function needing to send/receive data from another application (in a heterogeneous environment). It is a mandatory step that is usually done manually.

This problem is solved by a combination of two technologies/tools:

1. ASN.1: the ASN.1 standard consists not only in a textual notation for writing data types. It also comes with a set of **binary and textual encoding rules**. We only have to choose the kind of encoding we want (compact binary, XML, or a customized one) and the tools will generate automatically encoders and decoders for our data type.
2. Semantix tools: knowing that tool A is used to model the caller and tool B the callee, and knowing how tools A and B generate code, the tool will make the glue at code level between tools A and B data types and the ASN.1 encoder/decoder.

Encoded data end up in a buffer that maximum length is known in advance because it is calculated at interface view level. This whole process makes no dynamic memory allocation.

5.5 Glue code

Once all data conversion and encoding functions are generated, it was necessary to find the place in the code where to make the actual call to these functions. This required knowledge of the code generation strategy of each tool in use.

There are actually two different cases depending on whereas the interfaces are **immediate** (synchronous call) or **deferred** (asynchronous call). “Deferred calls” means that the caller sends a message and does not know about its delivery time. It is the most common semantics found in asynchronous state-machine oriented languages (such as SDL). “Synchronous calls” means that the call is done immediately and actually executed in the caller’s runtime environment. It is like calling a normal C function and waiting for the return before continuing, contrary to the deferred where it is calling a C procedure that has no effect on the caller (it just *informs* the operating system that the message *has to be sent*).

SCADE is a synchronous language that calculates outputs based on a set of inputs and an internal state. In that respect we can qualify SCADE operators, at code level, as **terminal functions**: they will never invoke other functions but only return some data.

On the other hand, SDL is an asynchronous language, based on state machines. You define a set of messages at model level (called “signals”) that can carry input parameters but not output parameters (as said above, output parameters are only possible for immediate invocation, which means terminal functions).

It is also possible in SDL to declare external procedures which correspond to the invocation of a terminal function. This is the way we have been able to integrate SCADE calls from within a state machine in SDL.

With this approach, we let the final user mix different languages with no particular restrictions, in a transparent way at model level. In the future, a co-simulation between the different tools is envisaged to be able to validate the system behavior.

At code level, things are also automated. The Semantix tool chain generates automatically the wrapper allowing the communication between SCADE and SDL language and between user code and the operating system.

In the future it is foreseen to support several other tools such as Rhapsody and Simulink. All the technology is in place and supporting a new tool is made extremely simple.

6 Conclusion and future work

This paper reports on a case study aiming at investigating the benefits of formal approaches applied to data definition and management in space software project. European space systems stakeholders are engaged in a set of studies and projects on these issues. The objectives are to solve the difficulties related to a proper integration of pieces of software from different providers (or generated from different generation tools and modeling environments). It is especially expected to ensure compatibility and correctness with as much rigor and as powerful proof and automation capabilities, than with formal methods and tools as applied to software functional behavior or non functional properties.

After a discussion of general issues related to space software data and their classification (section 2) we elaborated a formal model of data for a representative case study, encompassing their various characteristics (section 3). This data model not only demonstrated the capabilities of the selected formal data modeling language (ASN.1), but also served as a basis to practically evaluate the capability to automatically generate, on the one hand, contractual interface documents (section 4) and on the other hand, interfacing code to embed in the on-board software.

In addition to on-going work in particular to extend the covered data coding formats and software modeling languages and tools (e.g., Rhapsody, Simulink), future work is planned to evaluate the proposed approach from a global process point of view. Of particular interest in this perspective are the links with the space system databases. The standardization and formalization of data representation and the provision of support extraction tools are expected to improve both the efficiency and the quality of the associated software processes.

An additional track is the extension of the expressivity power of the modeling language in order to take account more functional properties such as the physical meaning of data. This extension would allow for instance avoiding the confusion between a data expressed in inches and a data expressed in meters.

Finally proposals must be submitted towards an evolution of the relevant engineering and validation standards applicable to space software, including the PUS standard (e.g., to separate the functional information and encoding rules).

References

Aho, V.A., Sethi, R., Ullman J.D. (1986): *Compilers: Principles, Techniques and Tools*, Addison-Wesley (1986)

ASN.1 information site: http://asn1.elibel.tm.fr

ASSERT: Automated Proof-based System and Software Engineering for Real-Time Applications, European Integrated Project IST-004033, FP6, 2004-2007.

PUS (Packet Utilization Standard) (2003): *Space Engineering — Ground systems and operations: Telemetry and telecommand packet utilization*, ECSS-E-70-41A, European Cooperation for Space Standardization, January 30, 2003

SSCDMT: System Software Co-engineering: Data Modeling Technologies: ESA/ESTEC Contract n°20467/06/NL/JD. 2007-2008.