Application of VDM-SL to the Development of the SPOT4 Programming Messages Generator

Armand PUCCETTI, Jean Yves TIXADOU

{armand.puccetti, jean-yves.tixadou}@cisi.fr

CS-SI
1 Av. Newton
92142 CLAMART Cedex
FRANCE

Abstract. This article describes our experience with the use of the formal specification language VDM-SL, in the evolutionary development of a safety critical system for the daily programming of the HRVIR (High Resolution Visible and Infra-Red instrument) payload of the SPOT4 satellite. This system controls the validity of the commercial photographic commands with respect to the constraints of the satellite and generates binary executable code that is then uploaded and executed on-board.

1 Introduction

This paper presents some intermediate results of ESSI project #27492 ISEPUMS. The aim of this project is to apply a knowledge acquisition method in combination with a specification one to a real-size application. The goal is to prove the validity of such approach and measure the expected gains. The application belongs to the area of Satellite Control: it is a sub-system of the ground segment of SPOT4.

The ISEPUMS project is actually at mid-term, so we will present results obtained so far, in terms of specification and method, and explain the work planned for the remaining time.

1.1 Aims and Objectives of ISEPUMS

In industries, many software projects still rely on semi-formal techniques, which are well mastered and whose cost can be predicted, such as V-shaped life cycles, iterative coding-and-testing, etc. Formal Methods (FM) provide means to develop reliable systems with fewer errors. But there are still obstacles to their wider use [BOWEN*]. Typical objections to using formal methods are: lack of maturity, lack of tools, difficulty of assimilating mathematical notations, and lack of experience. It is the latest point we address in ESSI project ISEPUMS. Its aim is to produce further

data on the application of FM to real problems in industry, to support them and bring these closer to assimilated and really practised tools. Another objective of this project is to bring this technology closer to the software engineers at CS-IS for better acceptance.

1.2 Plan of the Article

The following sections are structured as follows. Section 2 presents the context of the ISEPUMS, namely the Payload Management Centre (PMC), and describes the part of the PMC considered for rework. In section 3 we detail the approach taken for specifying the Programming Messages Generator (PMG) and describe its different layers. Section 4 gives the current status of the project.

2 SPOT4

SPOT4 is an earth observation satellite designed by CNES (French National Space Centre) and in service since March 1998. The general architecture of the satellite may be divided into the platform and the payload.

The *platform* provides the following functions:

- Interfacing with the launch vehicle,
- Power generation and storage,
- Satellite attitude, and orbit control,
- Management of housekeeping data,
- Telemetry downlinks, command uplinks and tracking,
- A stable 'optical bench' for the HRVIR and the Vegetation payloads.

The *payload* carries out the main mission. This consists of a payload equipment bay (PEB) and two HRVIR imaging instruments and Vegetation.

The PEB houses the electronics for image processing, recording and telemetry. Image data are compressed and formatted and then stored on the satellite, either by Magnetic Tape Recorders (MTR), or in a solid-state memory. The image data are either directly sent to ground stations or to the geostationary satellite ARTEMIS, if visibility allows, or otherwise sent later.

2.1 The Satellite's Management Centre

The SPOT4 positioning centre is used for monitoring and driving the satellite. The ancillary Systems Management Centre (SMC) is used to monitor the flight software and the equipment, to correct the attitude and orbit errors and to schedule the satellite's operations.

The development of the SMC was made by a team of CS-SI with an average of 80 engineers, and consumed 2500 men x months. The final code consists of 900 000 instructions of C code.

2.2 The Programming Messages Centre

The PMC centralises functionalities dealing with the payload. It guarantees the status of the satellite and is the only one that communicates with it for programming purposes.

A sub-system of the PMC is the Programming Messages Generator (PMG). It serves to program on a daily basis the equipment of the payload by means of commands, which can be interpreted on-board. Programs are made according to commercial photographic requirements given by the Commercial Ground Segment. The PMG controls the validity of the commands with respect to the constraints of the satellite and generates a binary executable code that is then uploaded and executed on-board. The PMG is similar to a compiler.

2.3 Re-developing the PMG using VDM-SL

The ISEPUMS project does a complete re-development of the PMG using formal techniques. We started with the same requirements documents [CNES*] and some expert on the PMC. The new PMG must produce identical output. Exhaustive benchmarks shall be executed to show identical results.

3 Re-specification of the PMG

This section details the methodology applied and the different components of the new PMG.

3.1 The FKRM Methodology

The Formal Knowledge Refinement Method (FKRM) is the combination of a Knowledge Acquisition (KA) method and a formal specification method. This method was developed within Euclid R.T.P. 6.3 project MOSES [MOSES97]. The aim was to develop a methodology suitable for complex military Knowledge Based Systems (KBS) projects.

The FKRM life-cycle is the combination of the following activities: knowledge acquisition from experts, specification of the acquired knowledge using the formal method, update of the knowledge base based on the formal specification and validation of the resulting specification by the experts, and further refinements.

The resulting product is a series of knowledge bases and associated formal specifications. Each specification is the refinement of the previous one.

Thus the complete refinement process can be fully traced even after the code generation if the formal specification tool allows this.

This method is based on a top-down approach. Since we have to start from an validated specification whilst being comparable with an existing implementation, a bottom-up approach is more adequate.

The FKRM method defines three phases in a software development process (see figure below).

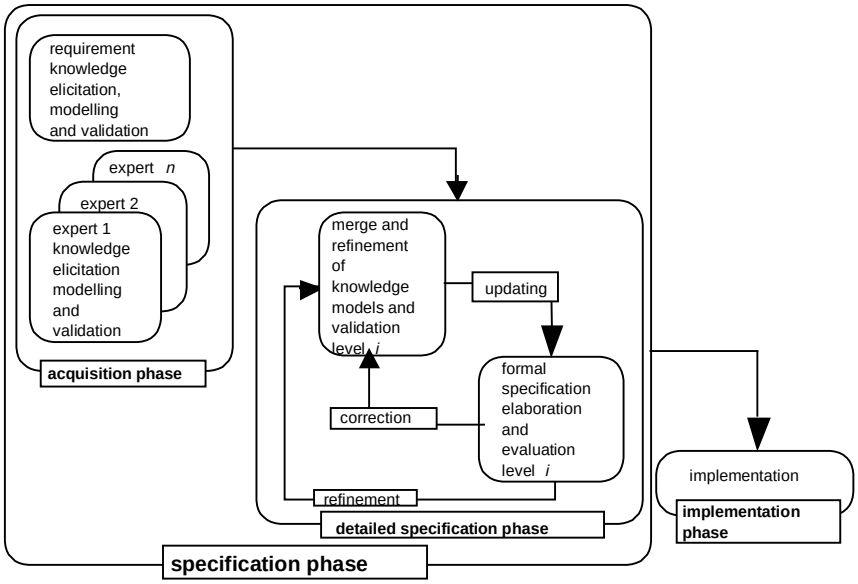
1- Phase 1 is a standard elicitation phase. The experts are listened to, for enumerating and analysing all the required knowledge from the technical domain as well as the application constraints.

This allows a first control of the consistency between experts and requires them to choose between different solutions or to find an agreement on conflicting points when necessary. The resulting Knowledge Base is then validated with the client.

2- Phase 2 is the specification phase where the knowledge acquisition method is combined with a formal specification method.

3- Phase 3 is the implementation phase beginning with the code generation and ending with the complete implementation of the system.

The most important and critical phase is the second one for it does the specification refinement. A refinement is made as follows. From the first level, defined in the global KB, one has to formalise the first KB. This task checks for completeness and consistency. Once the level 1 formal specification is completed, the corresponding KB shall be updated. The resulting KB is then validated. For the second level of refinement, more detailed information is added from the global KB. These tasks are iterated until all the global information has been used or the refinement reaches the maximum reasonable level.



1. The FKRM life cycle

3.2 The Application

Using the above methodology, the PMG was started with no a-prior knowledge of the domain. The layout of the tasks is as follows where tasks and sub-tasks written in italics are those already achieved:

Task 1: Understanding of the domain and the PMC application,
Task 2: Knowledge Acquisition:
Knowledge elicitation / acquisition of the PMG,
Validation of the knowledge model,
Task 3: Development of a parser of Payload Work Plan (PWP),
Task 4: Specification:
Definition of the VDM-SL data structures of the PMG,
Development of a VDM-SL interface layer,
Development of the PMG and prototyping:
Development of the Macro Commands (MC) code generator,
Development of the constraints checker,
Development of the Programming Message (PM) optimiser,
Development of the PM assembler
Testing,
C++ code generation,
Task 5: Integration and validation.

Task 2 provided a decomposition of the upper levels of the PMG, in terms of data objects, operations and triggers. The K.O.D.™ (standing for Knowledge Oriented Design) KA method [VOG88] has been applied. The expert was first interviewed to provide a high level description of the entities of the ground segment and a description of the functioning of the PMG. This information was then transcribed and analysed. The extracted objects were then structured into hierarchies of objects and the result validated by the expert.

3.2.1 Structure of the Application

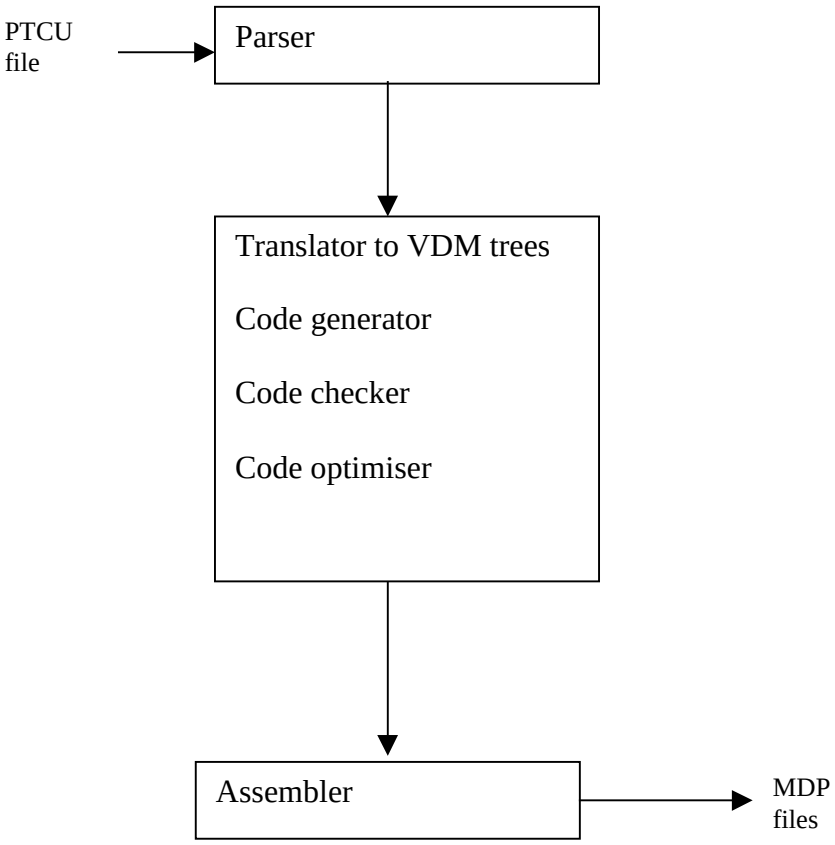
K.O.D.™ objects have been translated into VDM-SL: data objects and actions gave life to records and operations' signatures.

The VDM-SL code is completed in task 4, and the PMG consists of several layers of VDM-SL specification (centre box of the picture below). Included are modules to handle the following data types:

- VDM-SL format syntax trees,
- States and transitions,
- Macro-commands,
- Other data types.

The other boxes contain utilities interfacing the main modules with the environment.

Starting from a BNF syntax, a parser has been produced using the MOÏRA Toolbox [CISI98]. The parser is written in C and is interfaced to the main box via a Dynamic Link (DL) module [IFAD98].



2. Structure of the PMG

3.2.2 Description of the main VDM-SL Modules

Below, we go into further details about the main functionalities and data structures of the PMG.

The PW

Every day the OPMC prepares the work program for the next day. This yields a PWP, which is translated into a PM executable by the on-board flight software.

The PWP first parses the PMG into a tree, with the following type:

```
Tree = [leaf | node];
Leaf =     code*  -- Id. of the node
```

```
            data*
            tree*  -- First brother node if
            bool*  -- this flag is up,
            uid;   -- Unique identifier.

  Node =    code*  -- Id. of the node
            tree*  -- First son
            tree*  -- First brother node if
            bool*  -- this flag is up
            uid;   -- Unique identifier.
```

Sequences

A PM is structured into *sequences* corresponding to sequences of activities at the PWP level. A sequence leads the payload from a sleeping state to another sleeping state through a sequence of commands.

A sequence of PWP has the following type:

```
seq_data ::
       hb_begin   :nat        -– On-board start and
       hb_end     :nat        -- end date,
       nbb        :nat        -- Number of blocs,
       nbe        :nat        -- Number of states,
       all_blocs  :blocs;     -- List of blocs.
```

Whilst the commands are executed, the payload traverses different states using transitions. During a transition, MC (see below) are set to work to lead to a new state or change the satellite's configuration. The execution of a sequence of PM corresponds to a series of sequences of photographs.

Blocs

To exploit the satellite in a flexible manner, it is possible to program the characteristics of its two telemetry channels while doing a photograph on the other one. A photographic *bloc* is a sequence of photographs during which the snapshot or transmission mode is constant. Snapshot can either be transmitted to the ground or to ARTEMIS or recorded. A PWP also comprises memory reading blocs (of type `bloc_read`), mass storage management blocs (`bloc_mdm`), state description blocs (`bloc_state`) and memory dumping blocs (`bloc_dmp_mdm`). A memory reading bloc downloads the content of a tape recorder to a ground station. A state description bloc indicates an intermediate sleeping state between other blocs.

In VDM-SL this is modelled by

```
blocs = seq of bloc;
```

```
bloc = bloc_pdv|bloc_read|bloc_mdm
      |bloc_state|bloc_dmp_mdm;

bloc_pdv ::
            type_pdv    :pdv_types
            stations    :string10
            no_ems      :nat
            coding_id   :int
            hb_begin    :nat
            length      :nat
            pdvs        :seq_pdv
inv b == (b.no_ems in set {1,…,2});

seq_pdv = seq of pdv;

pdv ::
            HRVIR_no    :nat
            no_pdv      :nat
            hb_begin    :nat
            length      :nat
            mcv_pos     :nat
            mode        :mode_pdv
            gains       :nat4
inv p == (p.HRVIR_no in set {1,...,2}) and
            (p.mcv_pos in set {3,...,93});
…
```

Typically, the payload is activated by turning on successively the emitter and HRVIR and then performing a photographic bloc. Between each pair of photographs, the HRVIR's mirror can be repositioned and the emitter reconfigured. A transitions graph describes the possible states and transitions of the payload [CNES98].

The data type is trivially:

```
transition ::     start_state:state
                  end_state:state;

graph = set of transition
```

Macro-commands

The PMG produces a binary code ready for uploading to the satellite. The target assembly language has 16 instructions. These instructions are represented in the PMG as records, keeping only the useful parameters of each instruction and leaving out representation constraints.

Structure of modules

The main modules of the PMG do the following jobs:

- Translation of parsed trees: the syntax tree of the PWP is node-wise imported by the VDM-SL code, and high-level data structures elaborated,
- MC generation: the PWP is analysed and intermediate code is generated,
- Semantic checker: the PWP is checked for well formedness and for correct use of the satellite's resources. Finally a path through the PMG graph is generated,
- Assembly: generation of binary code from the intermediate MC code,
- Main module calling the previous modules in the correct order.

The core part of the PMG consists actually of 4200 lines of VDM-SL and 900 lines of interface code written in C and C++.

4 Status of the Project and future Plans

4.1 Current Status

The current specification contains the modules mentioned in §3.2. This set of modules can be executed for testing the correct production of transitions and macro commands. Input PWP files tested so far have about 1500 lines with approximately 100 photographs and 200 intermediate states.

4.2 Further Work

Future work includes:

- Validation of the VDM-SL specification by the CS/CNES expert: this is mandatory in FKRM and may lead to further corrections.
- Tests cases generation: further exhaustive test cases with detailed intermediate information will be produced.
- Code generation and testing: the IFAD Toolkit's code generator will be used to produce more efficient code.
- Comparison of the new PMG with the original one (in terms of errors per phase, efforts, etc) and feedback to the CNES and the UE.

5 Conclusions

The combination of a knowledge acquisition method and the use of VDM, with prototyping and multimodules capabilities, enabled us to model this complicated system and build a first complete executable specification in a reasonable time (5 months).

The use of a KA method quickly reduced the difficulties met in understanding and clarifying the structure of the PMG. The direct use of the VDM method to build a model has faced us immediately with the complexity and level of details of the PMG. Again, this turns out to be positive, as the domain expert was frequently called to clear up the ambiguous points. Solving these problems since the early stages will hopefully reveal to be beneficial in the later stages by reducing the number of errors in the code. The test tasks, to come, will provide some statistics in the near future.

Furthermore, VDM helps us to find the right choice in terms of abstractions, which results in a smaller system than the one obtained using classical techniques.

6 References

[BOWEN95a] Ten Commandments of Formal Methods, Jonathan Bowen and Mike Hinchey. IEEE Computer, 28(4):56-63, 1995.

[BOWEN95b] Seven More Myths of Formal Methods, Jonathan Bowen and Mike Hinchey (University of Cambridge). IEEE Software, 12(4):34-41, 1995.

[CNES98] CNES. Systèmes pour l'Observation de la Terre - SPOT4. Spécification des modes charge utile et des contraintes de programmation. Document CNES, 1998.

[CNES96] CNES. Systèmes pour l'Observation de la Terre - SPOT4. Spécification des messages de programmation charge utile. Document CNES, 1996.

[IFAD98] IFAD. Documentation package of VDM Toolbox V3.1.

[MOSES97] Knowledge Engineering for Advanced Military Information Processing. Final report of EUCLID R.T.P. 6.3, MOSES, 1997.

[VOG88] C. Vogel. Manuel K.O.D.™ CISI Ingéniérie, 1990.