Formal Design for Automatic Coding and Testing: The ESSI/SPACES Project

Eric Conquet[1] , Jean-Luc Marty[2]

Matra Marconi Space, Data Processing and On-Board SW Division,
31 Avenue des Cosmonautes, 31402 Toulouse Cedex 4 - France.
[1]Tel : +33 5 62 19 51 36
eric.conquet@tls.mms.fr
[2]Tel : +33 5 62 19 91 82
jean-luc.marty@tls.mms.fr

Abstract. Embedded software in the space domain must satisfy a set of strong constraints related to behaviour and performance, to fulfil user requirements. Moreover, due to the cost reduction trend in the domain and to the global necessity of increasing the quality of complex software systems, early design validation has become a real challenge for software designers. Currently used methods such as HOOD lacks support for behaviour description. Moreover, design validation is not feasible with such methods and first validation has to be made when the coding phase has sufficiently advanced. This occurs too late in the development phase especially when coding is essentially manual. This calls for the adoption of new development strategies based on formal description of the behaviour, on the use of simulation techniques to check the proposed design solution and on automatic code and tests generation techniques to increase productivity.
Following a preliminary successful experience in the context of an ESTEC R&D study called DDV(Dms Design Validation) [1] , the SDL and MSC languages and the ObjectGeode tool have been successfully applied on real projects. To complete those first applications of the technique, a PIE project (Process Improvement Experiment) in the ESSI program [2] has been proposed and accepted by the European Community. That project, called SPACES [3], aims at measuring improvement of development processes through the use of automatic coding and testing from a SDL model. This paper tells the complete story of SDL in our on-board division and focuses on the SPACES project and its current achieved results.

1 Introduction

For many years, development of software in different domains of industry has mainly focused on the choice and best use of programming languages. Then, to face with the increasing size and complexity of software systems, specification and design methods and tools have emerged. Their first purpose was to propose a common formalism, different from the programming language, to support the design process and ease the communication within the design team and with the customer. They helped very much especially on large projects with different contractors where a precise description of SW interfaces and SW role is mandatory.

J. Wing, J. Woodcock, J. Davies (Eds.): FM'99, Vol. I, LNCS 1708, pp. 57-75, 1999.

Although having been very helpful to support the design process, those methods have suffered from one major limitation : they were limited to the static description of the SW. Even if different methods promoted the extensive use of informal language to support the behaviour description, they were limited by the ambiguousness and lack of precision of such languages.

Then come the time of formal languages with their cortege of attractive concepts, hopes of powerful verification means and easiness of use. Needless to say that not every formal language fully meets those requirements. Especially when comes the time of industrial use which makes the availability of a powerful, robust and complete tool a must have if a real use on a project is foreseen.

Executable specifications and designs through the use of formal methods is very helpful during SW development since it eases early validations through simulations and automatic coding.

This paper tells the story of the successful use of SDL in the On-Board SW division of MMS through the very first experiments to the SPACES project, an ESSI PIE (Process Improvement Experiment) which is currently on-going and already gave interesting results.

- Section 1 describes the SDL language, how it has been selected for use on our projects, and its application on the VTC/SW project,
- Section 2 describes the current results obtained on different projects where SDL has been used,
- Section 3 presents the SPACES project, its objectives and development plan,
- Section 4 presents the results obtained so far in SPACES and what is expected from the next project phases,
- And finally, section 5 presents the expected impact of SPACES on our SW development processes.

2 SDL AND MSC at MMS: The Story Began with the VTC Experiment

2.1 Description of the SDL and MSC Languages and the ObjectGeode Product

SDL stands for Specification and Design Language. It originates from an IUT standard which evolves every four years. The 1992 release [4], which is now supported by the currently available tools, introduced the Object Oriented concepts into the language. Originally, the language has been defined to cover the specific needs of the telecommunication domain related to the specification and description of protocols [5] [6].

The SDL language is made of three different views :

- A hierarchical view where a root block constitutes the top of the hierarchy and includes other blocks. Terminal blocks are called processes since they contain the automaton which describes the behaviour.

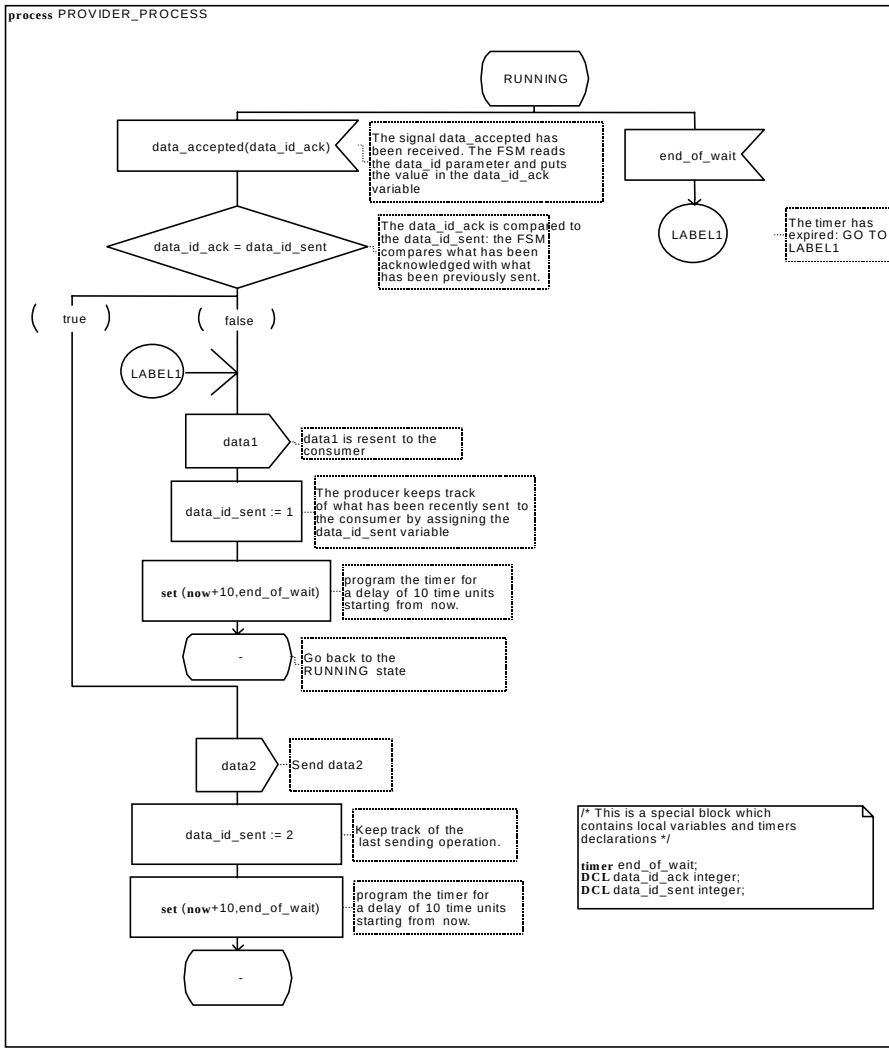

Fig. 1. A SDL automaton.

- A communication view where, for each level of the block hierarchy, a set of channels are defined connecting the blocks of the current level. Those channels carry signals which are used by the processes to communicate.
- An automaton view, where the behaviour of each process can be described. The automaton is made of transitions which can be fired when a specified signal has been received by the process. The figure 1 shows an example of such an automaton in SDL. It is sufficiently commented and should not require any additional explanation.

The MSC formalism is complementary to the SDL one. MSC stands for Message Sequence Charts and is used to describe the sequences of exchange between SDL entities. A set of MSC sheets can be combined with specific operators to define more complex scenario. Those operators express alternative, parallelism or sequence between MSC sheets. The figure 2 shows a simple example of such an MSC which describes a small sequence of interaction between two SDL blocks.

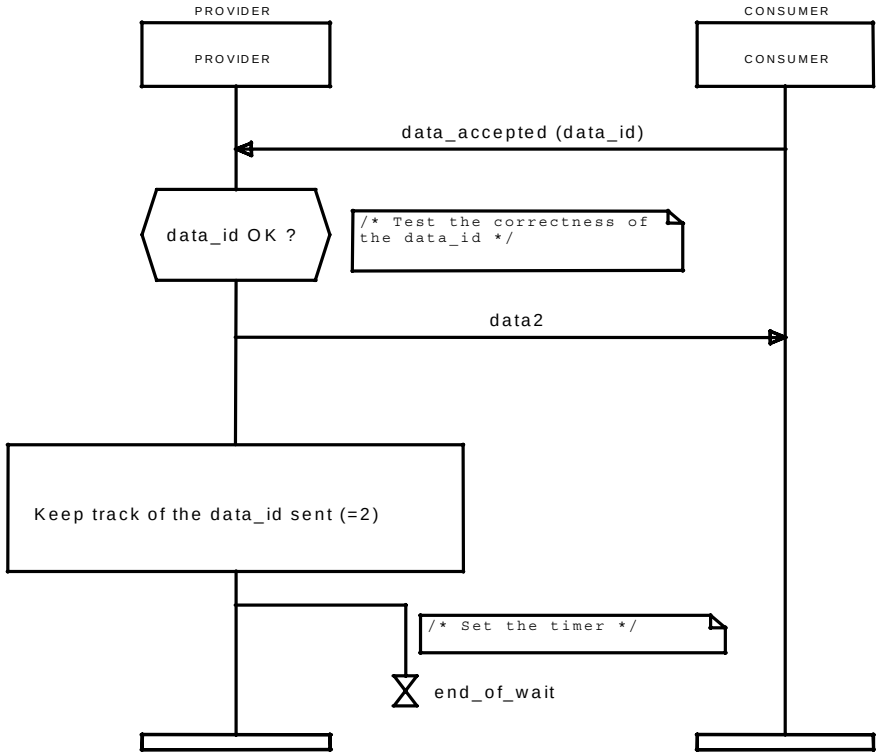

Fig. 2. A MSC example.

The ObjectGeode tool [7] is one of the tools which support the SDL/MSC formalism. It includes :

- A SDL graphical editor to build SDL models,
- A MSC graphical editor supporting the combination operators to build large scenarios,
- A SDL simulator which can use a set of MSCs as observers to verify the behaviour of the SDL model,
- An exhaustive simulator to verify model properties with a full analysis of model states,
- A document generator which supports different formats,
- A code generator which produces C code from the SDL model.

The figure 3 shows the different modules of the ObjectGeode tool.

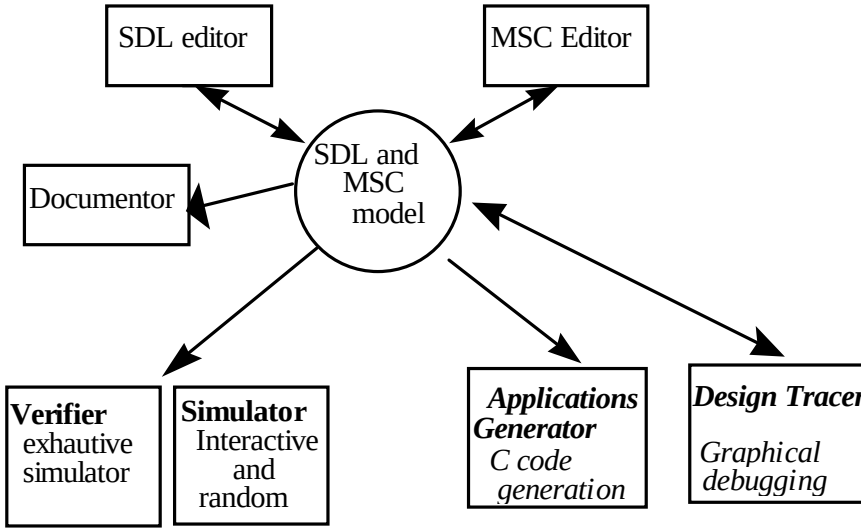

Fig. 3. Architecture of the ObjectGeode tool.

2.2 Selected for the VTC/SW Project

The DDV Project How SDL/ObjectGeode Has Been

MMS has studied the interest of modelling languages and simulations techniques for many years. Different techniques have been evaluated and applied on many projects mainly for performance evaluation of computer systems but more recently for analysing and verifying the behaviour of critical systems.

A R&D project, funded by ESA (European Space Agency), has been particularly fruitful in the selection process of a modelling environment which can be used on a real project for the purpose of modelling, documenting, verifying and sub-contracting a SW specification and design. This project called DDV (DMS Design Validation) [8] began in 1995 and analysed a large set of modelling formalisms with the purpose of selecting the most appropriate one for modelling and verifying a DMS architecture in the specification and design phase. The selection phase, led with a strong support of the VERIMAG laboratory in Grenoble, France, concluded on the appropriateness of the SDL formalism and on the maturity of the commercial tools.

The formalism has been applied during the specification phase where we modelled the architecture of an avionics system. Both the hardware architecture and the DMS layer were modelled and a set of properties were verified with the exhaustive simulator of ObjectGeode. Then, the specification model has been delivered to the designer team (Dornier Satelliten System Gmbh) who refined the model and verified the same properties. The global conclusions of the study were very positive and convinced MMS to apply the formalism on a real project.

The VTC/SW of the COF/DMS Was a Good Candidate

MMS is in charge of the development of the DMS (Data Management System) of the COF (Columbus Orbital Facility) in the ISS (International Space Station) project. The On-board Software division of MMS is in charge of developing two main SW sub-systems in the DMS: the VTC SW (Vital Telemetry/Telecommand Controller) and the MSW (Master Support SW).

VTC is a corner stone of the Columbus Orbital Facility (COF). The vital layer is in charge of managing vital equipments, reporting to the Space Station, processing commands, starting and monitoring the nominal layer. The MSW is responsible for the different on-board experiences and as such must provide facilities for the scientists to operate these experiences.

VTC is made of two redundant computers in master/slave configuration. Both computer acquire sensors data through 1553 buses and direct HW interfaces, monitor them locally and transmit results to the space station through Telemetry frames. A set of Telecommands can be processed by the VTC for the purpose of configuring equipments or the acquisition/monitoring process. The status of the nominal layer is permanently watched by the VTC and appropriate reconfiguration processes can be started if the VTC detects an anomaly.

The VTC project has been for us a very good opportunity to evaluate the technology on a real project. Firstly, it was large enough to produce meaningful results which could be used to convince other teams of the interest of the technology. Secondly, it is a critical part of the DMS in charge of controlling the vital layer of the COF. As such, it has been classified by ESA as a category B SW. This indicates that the SW contains a set of mechanisms on which the safety of the COF relies. Consequently, there is the opportunity to analyse and validate the behaviour of some critical mechanisms in the SW, which is just why formal languages have been defined. Thirdly, the Columbus standards, taking into account the critical dimension of the SW, proposed to complement the HOOD methodology with a formalism addressing the objectives of behaviour modelling and verification. The selected formalism was Petri Nets for which a coupled use with HOOD was described. Having no experience with Petri, and considering that the commercial tools when they existed were not mature enough, MMS proposed to DASA (the COF project manager) to replace Petri with SDL supported by the ObjectGeode tool. That proposal has been accepted after some needed clarifications from MMS. And, last but not least, the project team was really motivated by the experience which reveals a real mind openness to new technologies in our division.

This has been the first stone of the first industrial use of SDL at MMS and this is the story that we are going to tell in the rest of the paper.

2.3 Use of SDL on the VTC Project

First Step : Which Modelling Strategy to Apply ?

Although a standard existed preliminary to the decision to use SDL, it was appropriate to Petri Nets and proposed only a limited use of the modelling technique. We have considered that our knowledge and experience with SDL could be a good basis for a larger experimentation of the technology.

The selection of a modelling language and tool is just the first step in the elaboration of a methodology which can guarantee satisfying results. Formalisms such

as SDL and MSC, and the facilities provided by the ObjectGeode tool, can be applied on different types of projects, with different goals, to provide different kinds of results.

A very light use of those technologies may be sufficient for either small projects or projects for which a full use of a modelling and simulation technology would be oversized with respect to the real needs. For such projects, formalisms such as SDL and MSC can be used only for documentation purposes by focusing on the visual and legible aspects of their graphical representations.

Another possible use is to combine the visual dimension of the formalism with the power of interactive simulation in order to run different scenarios and to produce traces under the form of MSCs. This strategy is interesting when the global behaviour of a system must be understood and analysed to demonstrate that the proposed architecture fulfils the user requirements.

A more formal approach consists in using the formalism, and the tool features, to prove that a given set of properties is fulfilled by the model. Exhaustive simulation is the ideal candidate for such purpose but although being very performing it has several limitations and cannot be considered as a push-button verification strategy. First of all, due to the nature of the technology itself (exhaustive simulation means a complete exploration of all the system states), the size of the model must be reasonable, which generally means small. Secondly, before verifying properties, they have to be written by using either the same formalism as the one used for modelling, or another one more dedicated to the task. There is a true properties modelling activity which requires the same level of attention as the system modelling.

In the case of VTC, we've decided to adopt the second modelling strategy described above : a complete modelling of the SW architecture and a set of verifications by using interactive and random simulation.

SDL and MSC with HOOD in the VTC/SW Project

One major characteristic of the SDL experience in VTC is that the HOOD methodology has been kept as the main design method. This introduced some constraints on the use of SDL but gave us interesting results on the comparison between the two methods.

The experiment [9] took place during the Architectural Design phase of the project where the whole software architecture is designed. The experience consisted in four majors phases as depicted in figure 4:

- A complete modelling of the SW functions with MSC diagrams,
- A complete modelling of the SW architecture with SDL,
- A simulation of the system functions and analysis of the SW behaviour with the ObjectGeode simulator,
- A generation of the complete MSC traces produced by the simulator after execution of the system functions. Those traces should be used to ease the elaboration of the integration test plan.

In the first phase of the experiment, each function of the system, the cyclic activities and the asynchronous operations (TC processing), have been described with the MSC formalism. Due to the complexity of the behaviour, and to the size of interactions sequences between the different SW entities, the scenarios have been broken down into small MSC leaves which have been organised into larger scenarios with the combination operators. The elaboration of MSCs has been performed at each successive level of the HOOD architectural design, to provide a full traceability between the system functions and the use of HOOD operations visible at each level.

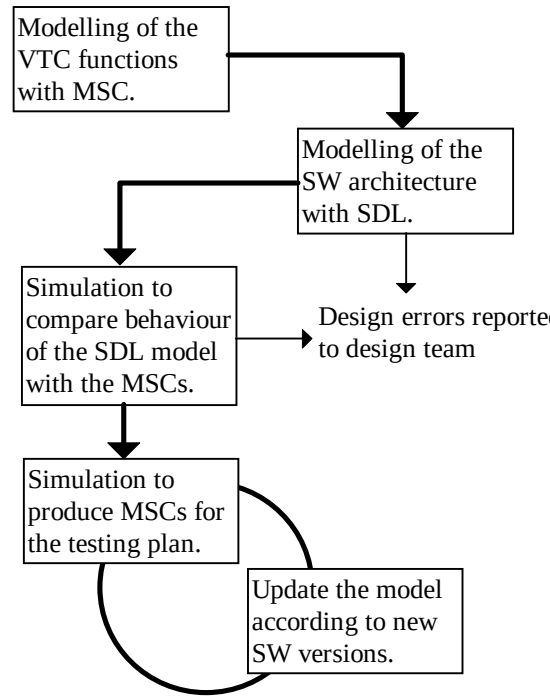

Fig. 4. The SDL experiment on VTC.

In the second phase, the complete architectural design of VTC/SW has been modelled. Although only the dynamic behaviour of the application was concerned by the modelling, it could have been sufficient to restrict the model to only terminal and active objects. We have considered that the model should also be a complete representation of the HOOD design with a more precise and complete description of the behaviour. For those reasons, the SDL model is structured just like the HOOD design, it contains only the active objects and limits the level of details to those absolutely needed for modelling behaviour. For example, details of data structures have not been represented when they were not essential to fulfil the modelling/simulation objectives.

In the third phase, having a detailed model in SDL and a description of the functions to be performed by the system, in MSC, we were ready to start a simulation campaign for demonstrating the correct behaviour of the proposed design. The MSCs which have been initially written for documentation purposes have been lightly rewritten to satisfy the constraints imposed by the simulator. The model has been compiled with each interesting MSC and the simulator used for verifying that the behaviour described in the MSC was exhibited by the model. If not, a detailed scenario was automatically generated which gave the complete sequence of interactions leading to a discrepancy between the actual behaviour and the expected one. Such a process perfectly worked even with very complex MSCs made of different sequences combined with operators. Such a capability of the ObjectGeode tool considerably reduced the time spent in the verification phase.

Those three above steps already formed the basis of a very positive experience. And, in fact, that experience should have stop here. But, without anticipating on the

conclusions, we can say that the experience has gone further than initially scheduled mainly because of the positive results it gave and the increased interest of the project team in the technology. The last step consisted in the generation of the MSC traces for each function performed by the VTC/SW and modelled in SDL. One major interest of such traces is that they give, in a very simple and easy-to-read form, a complete description of the interactions between SW entities (Objects or tasks) which occur when a given function is executed. Those traces are very helpful for documentation purpose, especially when new designers join the team or during the maintenance phase. Moreover, they can also be used for elaborating the test plan for the integration phase since they contain all the requested details.

3 The VTC/SW Successful Experiment Convinced New Project

3.1 Major Conclusions of the VTC/SW Experiment

A Positive Experiment

If the challenge is to use the smallest number of words as possible for summarising the conclusions after the experiment, we would say « it is a very positive experience ». To say more, we can list here above the major conclusions :

- SDL and MSC are visual and formal languages which make them rather easy to read and non ambiguous.
- SDL and MSC are not very complex to use. They can be easily manipulated by any SW designers. This is a major advantage compared to other formal languages.
- SDL and MSC significantly improved the communication within the design team, especially when new members joined the group. This is specially due to the great legibility of the formalisms. The use, by the designers, of the SDL model to write the ADA pseudo code is a good demonstration of the real legibility of the formalism.
- The SDL modelling activity provided interesting feedback to the designers and helped in identifying incompleteness and inconsistencies in the HOOD design. This shows that elaborating an executable model with a formal language forces the designer to enter a higher level of detail and helps him to trap design errors earlier.
- The MSC descriptions of VTC functions has been found very clear and precise and the SDL model has greatly emphasised the description of the tasks, especially from the behavioural point of view.
- The level of confidence on the quality of the SW design has been increased according to all project members. This is due to both the results of the analysis phase with SDL, where inconsistencies have been found, and to the verification phase where all the nominal behaviours, as described with MSC diagrams, have been checked with the simulator.
- The final customers (ESA and DASA) have very much appreciated the content of the Architectural Design Document, which contained the SDL model. It

allowed them to more deeply understand the architecture of the VTC/SW, especially from the behavioural point of view.

- The SDL model, initially elaborated for the Architectural Design Phase, has been continuously maintained since its very first version and is now completely in phase with the current release of the SW. This is a direct consequence of the positive conclusions above described and it is a decision of the project manager to ease the integration testing phase with the help of MSC diagrams generated by simulation.

Some Limitations Found

Although having been very positive, the VTC/SW experiment raised some limitations in the domain covered by the tool or some difficulties in the introduction of the technology in our processes.

- The SDL/MSC formalisms are much appropriate for modelling the SW behaviour, but when performance or Real Time behaviour of the target must be analysed, some limitations appear. The formalisms, and the supporting tool, are not appropriate for such tasks, even if it is always possible to model HW resources or operating system but at the price of a substantial model modification. This has been a major limitation during the experiment since performance of the SW was also a major concern. Consequently, there is here a place for SW gateways between SDL tools and true performance evaluation tools such as OPNET.
- The major problem of any modelling activity is the selection of the right modelling level. Obviously, not all the system features can be modelled, and the model must be adapted to the real needs. This point has to be discussed before the modelling activity starts.
- Our experiment has taken the HOOD design as the main driver of the modelling activity, and most of the HOOD model has been translated into SDL. The mapping has been quite easy to establish due to the closeness of concepts from the two methods. As a result, there are many redundancies between the SDL model and the HOOD design. This will lead us to deeply investigate the problem of using SDL from the methodological point to view, and to study how a smooth transition from HOOD to SDL for example can be put into practice.

3.2 Applications of SDL/MSC after the Successful VTC Experiment

Considering the very positive results we obtained on the VTC/SW project, we have decided to enter a new phase in the integration of the SDL technology in our SW development process. It has been concretised by different types of actions : application of SDL to existing projects, deeper analysis of new features of the ObjectGeode tool and a more general evaluation of the potential impact of the technology on our processes.

New Applications of SDL

After the VTC experiment, which is still running to maintain the model up to date, a new project has been selected to use the SDL and MSC formalisms. It is the SPOT5/HELIOS2 Central Flight Software which was also designed in HOOD. But

the application to this project has been very different from the first experiment on VTC/SW. As most of the code was reused from the SPOT4/HELIOS1 version, it has been considered as not mandatory to build a SDL model of the SW but to focus more on MSC for improving the description of the dynamic behaviour of the SW. For that reason, only some parts of the SW has been selected, generally those for which the dynamic behaviour was complex to describe. The results on that project have also been very positive, for the project team as well as from the customer point of view. This project confirms that those modelling technologies can be easily adapted to the project needs.

Following that successful experiences, another projects decided to use the SDL/MSC languages for different purposes.

- On ARIANE V project, we decided to use SDL and MSC to monitor and master the re-design of the IMU (Inertial Measurement Unit) SW of the launcher. The specification and design activities are performed outside MMS by Sextant Avionique. We expect from the use of SDL to better understand the proposed design and to improve our role of SW architect. This is an on-going project and the first conclusions already showed that some inconsistencies in the informal specification can be found by modelling it with SDL.
- On the MSW (another part of the COF Data Management System) we have used MSCs to describe tests sequences to be executed during integration phase. MSCs are easy to build and to understand, and ease the understanding of the SW dynamic, especially for some complex behaviours.
- We are also trying to convince the project to use simulation to verify the design and code generation to generate a prototype of the application able to run on VXWORKS targets.

Entering SPACES

Those on-going use of the SDL/MSC formalisms and ObjectGeode tool are concrete applications of our development processes which recommend those languages to support, when needed, the SW development phases. The SDL/MSC and ObjectGeode features whose interest has been demonstrated are recommended for use on real projects. As process improvement is a continuous activity, always aiming at defining better processes, there is clearly a need to investigate unexplored features of a given technology.

In the ObjectGeode tool there are at least two features which seems interesting: the code generator and the capacity of producing test sequences. The past and current applications on real projects did not use all of the ObjectGeode features, either because they were not really needed, or our experience with them was not sufficient to start a real use on a project. To complete our knowledge of the real tool capabilities, we sent a proposal to the European Commission in the framework of the ESSI program (European System and Software Initiative). That proposal, called SPACES (Software Production using Automatic Code Generation for Embedded Systems) has been selected for funding by the commission and started in November 98.

4 The SPACES Project

SPACES is a PIE project according to the ESSI program. A PIE (Process Improvement Experiment) "provides the opportunity for an organisation to demonstrate the benefits of process improvement, through a controlled, limited experiment. The PIE is undertaken in conjunction with a real software project (the baseline project) which forms part of the normal business of the organisation" according to the ESSI web site [2]

"A PIE allows an organisation to try out new procedures, new technologies and organisational changes before taking the decision as to whether or not they should be replicated throughout the software developing unit" [2].

According to that definition, a PIE project must:

1. Define process Improvement Objectives from business and technical point of views,
2. Select a set of limited changes whose impact have to be evaluated with respect to the selected objectives,
3. Choose a baseline project on which the experiment will be performed. A PIE is not a complementary funding source for the baseline project but a mean for evaluating the interest of a new development strategy which is applied on a real project.

4.1 The SPACES Objectives and Development Plan

One of the major objective of the SPACES project is to demonstrate if code and test generation techniques can significantly increase our productivity and reduce the time to market of our space systems. This objective is mainly business oriented since the introduction of a new technology in a development process has an interest for an industrial company if it can significantly improve its capacity to develop its core business and its market share. In the SPACES project, we targeted a reduction of almost 30% of the SW development effort by using automatic code and test generation techniques. Moreover, by using those techniques, we expect a significant increase of the confidence level that we and our customer have in our systems.

Naturally, this business objective is directly related to technical ones. Proving that those techniques can increase the productivity of our teams implies to demonstrate that they are really applicable in our projects. The main question from the technical point of view is the following: are we able to produce real code for on-board applications with real-time constraints?

Four Phases in SPACES

According to those objectives, the SPACES project is made of four major phases:

- Apply code generator to the VTC model and analyse the qualities of the generated code. This requires the addition of design details in the model, specially at data and algorithmic levels which are very simplified in the model. The quality of the generated code is evaluated from the quality standards point of view and also with respect to performance. The time to produce the final code is also measured and compared with the figures from the baseline project.

- Optimise the generated code. The first versions of the generated code will probably not match the quality standards we expect for embedded SW. As direct modifications of the generated code must be avoided, there are two different kinds of solutions. The first one consists in defining modelling rules at SDL level which forbid the use of some SDL statements whose translation into code is not optimal. The second one consists in modifying the code generator itself by providing new translation rules for some SDL statements. Technically speaking, the second solution is totally feasible thanks to a real openness of the ObjectGeode code generator. But it must be performed carefully since the new code generator can violate SDL semantic.
- Produce automatically tests sequences and generate the testing monitors. Testing is generally a painful task. First, the tests sequences must be defined from the testing objectives. Then, they must be executed on the target and their results analysed. This process is generally manual and consumes time and money. Our purpose in SPACES is to demonstrate that appropriate test sequences can be automatically generated from a model. Given a test objective, the ObjectGeode simulator is used to produce the test sequences which match the test objective. Those sequences are described as MSC diagrams to document the testing activity. And they are also used to produce the testing monitors which will capture data from the running code and verify if the test passed.
- Perform cross-testing on the final target. Our main goal in this task is to compare the coverage level of automatic and manual tests sequences. The manual code will be tested with the automatic tests sequences, and the automatic code will be tested with both manual and automatic tests sequences. By comparing the number of errors found in each case, we will have an idea of the respective quality of the tests sequences and the codes. By crossing those figures with the time spent on setting up each testing strategy, we can estimate the gain in terms of productivity to reach a given quality level.

5 SPACES: Already Positive Results and Other Expected

5.1 First Results from SPACES Now Available on Earth

At the present time, the SPACES project has almost reached the end its first phase and some results already exist.

From Model to Code in an Incremental Way

- The list of discrepancies between the model and the existing ADA code has been produced. For each of those discrepancies, a solution has been defined to update the model.
- An incremental approach has been set up to evaluate the code generation technique. Three versions of generated code have been defined as shown on Figure 5: V0 which is the initial version of the model, V1 which integrates some facilities for entering TCs and which also produces TM flows in the correct

format, and finally V2 which will improve the TC interface by including decoding mechanisms which take a CCSDS message on input.

Successful Code Generation on Various Platforms

- The version V0 has been generated and executed on UNIX platform. No problem have been found during this first trial. Thanks to the structure of the code generator, generating code from a SDL model on any of the supported platform is really a push-button activity. The behaviour of the generated code is identical to the model1 and this can be checked either manually through traces produced by the code, or more easily with MSC and SDL model animations generated at run time and processed by the Design Tracer tool.

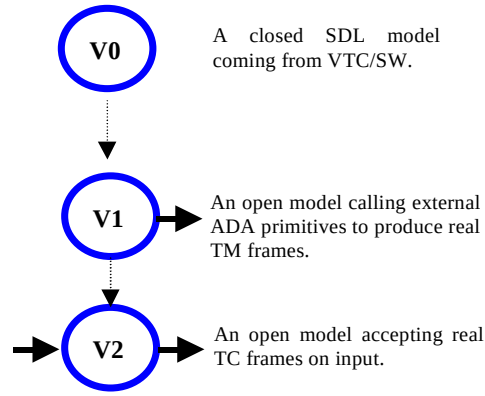

Fig. 5. The different produced versions of the generated code.

- To demonstrate the capability of the code generation, we have also try to generate and execute the code on another platform. The test has been performed on Windows NT platforms, and once again, a simple push-button action has been sufficient to produce the code which exhibited the same behaviour as on Unix.
- The V0 version has also been ported to a true real-time target: a Force Computer board with the VXWORKS operating system. Porting has not been very difficult since the code generator is already customised for that environment. The generated code run perfectly the first time it was loaded on the target. Moreover, thanks to a complete integration between ObjectGeode and the Tornado development environment, debugging at SDL level is possible and really operational. Debugging trace at code level are used to locate SDL statements in the model, and breakpoints can be put directly in the SDL model through SDL editor.

Subsequent Versions to Be Closer to Real Targets

- The production of the V1 version required use of external ADA code which implement primitives used to build telemetry packets from internal data. That code is produced automatically by the Production Program which takes on input information coming from the system data base and generate the primitives.

[1] There are some differences due to some SDL concepts which are nice from the theory point of view but not realistic. This concern especially the way time is managed in SDL.

Modelling this mechanism with SDL is not very appropriate, so we decide to couple the C code generated by ObjectGeode with the ADA code generated by the Production Program.

- For what concerns the V2 version, it has been completely developed and successfully tested on Unix platforms. That version introduces an interesting feature by directly processing real TC messages (CCSDS frames) instead of abstract representations through SDL signals. It means that the produced code can be easily integrated with real hardware and specially with bus controllers from which it can get the CCSDS frames.
- Our experience also showed that integrating external code at simulation level was almost as easy than with generated code. This is an interesting result since it will allow our development teams to perform early design generation by introducing in an incremental way external code.

Success in Code Generation Impacts the Reuse Policy

- For now we can consider that code production through automatic generators has really been demonstrated at least for prototypes. For what concerns the possible use for embedded applications, a deeper analysis on the quality, conformity to the standards, performance, and size of the generated code has to be performed. Some preliminary analysis showed that with easy to use optimisation techniques, code size can be dramatically decreased (30% gained by a very simple optimisation rule: precise for each SDL process that only one instance is authorised at execution time).
- For what concerns the productivity issues, it is clear that the code generation task, with or without integration of external code, is not time consuming. One day at most has been necessary to get a code whose behaviour has been checked with respect to the original model. Moreover, changing the target platform, or even the system architecture, by introducing multiple processors, is not really a major problem, since the code generator already addresses those issues.
- The code generator integrated in ObjectGeode only produces C code when ADA is preferred and imposed for embedded systems. Our analysis show that the target language has least importance when code generation is automatic. In classical development strategies, the assembly code is merely analysed or modified (unless performance optimisation are needed or when parts of code are directly written in that language). Constraints which exist at ADA level, and which are verified by the compiler, can be put at SDL level instead.
- From the reuse point of view, we think that a combined use of SDL modelling and automatic code generation techniques can improve a lot the reuse process. If reuse is limited to piece of codes, it cannot be optimal since code is generally very much related to a given target. If reuse takes place at design level, complete parts of SDL model can be reused and combined to produce new models which can be verified through simulation and used to produce new implementations for other targets or even different architectures.

5.2 SPACES: Second Stage Ready for Ignition

Ready for Testing

The second stages of the SPACES project is now ready for ignition and will start as soon as the first stage is completely finished. First results obtained during first stage operational phase are very promising for the project's future and at the time this paper will be presented, we can expect many positive conclusions.

The second stage will address two important issues:

- Automatic generation of tests sequences by model analysis.
- Automatic generation of tests procedures to monitor the target code.

SPACES aims at reducing the duration of the testing phase and improving its efficiency. There are three major problems during testing: find the most accurate tests sequences, write the test procedures and apply them to the real code. Testing is critical but generally painful and it is certainly the place where productivity can be most easily increased.

Verification and Validation Testing

As an exhaustive test of the code is generally not feasible, appropriate tests sequences have to be selected trying to find all potential or real bugs. There are two kinds of tests (not to be mixed with testing levels: unit and integration): verification and validation. Verification testing aims at proving that the SW design does not introduce improper behaviours leading to code crash, validation testing aims at proving the SW fulfils its specification.

We think that the use of a formal model, and SDL is a formal language, can help in elaborating those tests sequences. For what concern verification testing, the knowledge of the model architecture and of the code structure, should be sufficient to produce tests sequences by techniques of model analysis. The purpose of those tests sequences is to test most of the possible behaviours of the code as identified through the model. Validation testing requires the knowledge of the tests objectives and of the model and generated code structure. Tests objectives cannot be automatically derived at validation level, and it is not desirable for them to be derived since tests would be defined by the same tool which produce the code. Those objectives must be given by the testing team and derived from the SW specification. Use of MSC, SDL, logical assertions or GOAL[2] models are possible means for defining such objectives and the list is certainly not exhaustive. By using those high level objectives and mastering SDL model and generated code, tests sequences matching the objectives can be derived.

Another painful activity in testing is writing tests sequences which have to be executed on the target. Use of a dedicated testing language or a classical programming one does not significantly change the nature of problem. Our solution consists in starting from tests sequences, written in MSC for example and produced either manually or automatically, generate the testing code and execute it automatically either on the target or on a testing platform which monitors the target. The activity of producing appropriate test code is then significantly reduced and testing can be almost made automatically.

---

[2] GOAL is a language used in the ObjectGeode tool to define model observers.

The SPACES project is an experiment of those techniques and has to prove their interest and efficiency on real projects. For testing issues we have to demonstrate that the solutions described above can be put into practice at a reasonable cost and can save time and money with keeping the same quality level or reaching a higher level. Especially, we would like to demonstrate that tests sequences generated from the model, either at verification or validation phase, are at least as efficient than the manually produced one to capture bugs. To verify that we will apply the automatic tests sequences both on manual and on automatic code.

6 What Do Exist after SPACES ? A New Development Process

The SPACES project has already produced some interesting results by demonstrating that executable code can be easily generated from a SDL model. A more detailed analysis has to be performed to confirm use of code generation techniques for embedded Software. But the main challenge of SPACES is not technical but organisational. Proving that a method, a technique or a tool is really powerful is one thing, but making it efficient on real projects is really a matter of organisation. The focus must be put on software development processes: how must they evolve to integrate a new technology? This is specially hard for code generation technique since it is not a simple box replacing manual coding. Introduction of such techniques requires important modifications of the whole process, especially during the design phase where more time shall be spent on software modelling.

For what concerns the specification and design phases, SDL has shown that it can be used for modelling a complete SW design, verifiable by simulation and usable for producing code able to run on a final target. But SDL has also some limitations, especially when data and algorithms have to be defined. The ObjectGeode method guidelines proposes an integration with OMT/UML class diagrams to address that issue.

The SPACES experience raised the problem of the evolution from HOOD to new design methodologies such as UML which is now arriving with a lot of promises. Due to the very recent release of a stable version of the UML notation [10], there is no available tool fully supporting it. But there are a lot of plans to provide new generation of tools which will implement several views of the notation, and some of them will be more dedicated to real-time systems [11]. VERILOG has such plans to provide a UML tool using the SDL and MSC notation to describe the behaviour.

It is clear that a full use of SDL to support the design phase will completely change the look of that phase. More time will be spent in it to validate the design and to produce executable prototypes on development or targets platforms. The development will be more incremental but at each design step, the implementation can be quickly verified.

During testing phase, we really think productivity can be increased with the same quality level. Automatic production of tests suites directly from the model, and automatic execution on the target will significantly reduced time wasted in painful task, giving more time to execute more tests.

7 Conclusion

People sometimes think that modelling tools require few efforts and produce a lot of interesting results that no human can produce. If the tool does not satisfy those requirements, it is not a good tool and investment will not be profitable. Those miracle tools do not exist and ObjectGeode is not an exception. But with the appropriate level of project and people investment, those tools can help a lot in the documentation of the project, in the communication within the team and with the customer, and are very helpful for early validation of a SW architecture.

The SPACES project shows that, not only formal methods such as SDL are powerful enough to almost completely support the SW design phase, but also that with proper use of powerful tools such as ObjectGeode, code can be produced and easily ported on various platforms. Moreover, even if the project did not prove yet that generated code is ready for direct integration in embedded systems, it has proven that working prototypes integrating external code can be automatically produced. If SPACES also proves that the testing phase can be improved, we will consider that ESSI experiment as fully successful. If the results are not fully satisfactory, the experiment will help us in defining rules for successfully applying the technology, taking its limit into account.

We are now convinced that such techniques, and their integration in the UML world, constitute the basis of new SW design processes. They will surely increase the productivity of the designers and significantly improve the confidence level that system providers and customers can reasonably put into the delivered products. We have now to be prepared to improve our development processes to be more cost effective and more competitive and those modelling techniques are one of the key of our success.

8 References

1. E. Conquet, Ph Humbert, V. Debus, J. Sifakis - Data Management System Design Validation - Final report - 09/96 - ESA contract N° 9558/91/NL/JG, WO N°20.

2. European System and Software Initiative -web site: www.cordis.lu/esprit/src/stessi.htm.

3. Matra Marconi Space - SPACES, Software Production through Automatic Coding for Embedded Systems. Project Programme V2.1 – 08/98.

4. ITU-T, Recommendation Z.100, Specification and Description language (SDL), COM X-R 17-E, Geneva, March 1992.

5. Systems Engineering Using SDL-92 , A. Olsen, O. Færgemand, B. Møller-Pedersen, R. Reed and J. R. W. Smith. North-Holland 1994, ISBN 0 444 898727.

6. SDL - formal object-oriented language for communicating systems, J. Ellsberger, D. Hogrefe and A. Sarma. Prentice Hall 1997, ISBN0-13-621384-7.

7. VERILOG, ObjectGeode method guidelines, version 1.0 – VERILOG - 1996.

8. S. Ayache, E. Conquet, Ph. Humbert – Specification and early validation of autonomous spacecraft fault tolerance using SDL – DASIA 96 – Roma, May 1996

9. E. Conquet, G. Touet - « Modélisation et validation d'un logiciel critique en phase de conception, une expérience d'utilisation de LDS/ObjectGeode » - EC/NT/LB/97.092 - 01/98.

10. UML Notation Guide version 1.1, Rational Corp, et.al., OMG,
Sept. 1997

11. Doing Hard Time, Bruce Powel Douglass, Addison Wesley, 1998,
ISBN 0-201-49837-5