Experiences with the GENE-AUTO Code Generator in the Aerospace Industry

A.-E. Rugina[1], J.-C. Dalbin[2]

1: Astrium Satellites, 31 rue des Cosmonautes, 31042 Toulouse cedex 4
ana-elena.rugina@astrium.eads.net
Phone: +33 (0)5 62 19 78 06
Fax: +33 (0)5 62 19 71 58

2: Airbus Operations SAS, 316 route de Bayonne, 31060 Toulouse cedex 9
jean-charles.dalbin@airbus.com
Phone: +33 (0)5 61 18 08 76
Fax: +33 (0)5 61 93 03 54

Abstract: This paper gives an overview of the most recent experimentations that Astrium and Airbus conducted with the GENE-AUTO code generator during 2009. GENE-AUTO is an open source automatic and qualifiable C code generator taking as input Simulink®/Stateflow® and Scilab/Scicos models. It was developed in the context of an ITEA European project that ended in December 2008 (www.geneauto.org). The GENE-AUTO toolset is currently maintained by its developers and evaluated for industrial usage by several end-users. This paper presents the case studies that we used for evaluation purposes, explains the organisation between the users and technology providers with respect to the toolset maintenance and summarizes the experimentation results.

Keywords: qualifiable code generator, Simulink®

1. Introduction

The development of real-time embedded systems, such as those in the aerospace domain, faces well-known increasing constraints, in particular related to higher system complexity and shorter time to market. Such conflicting requirements are the origin of significant efforts towards the improvement of the current development processes. In the aerospace industry, increasing complexity is due to increasing functional requirements as well as related to autonomy, safety, performance and dependability. This complexity maps to a growing number of software functions that must be thoroughly tested and compliant with the certification standards (DO178B for aeronautics, ECSS for the space domain). In addition, in the space industry, the system must be designed to favour maintainability during system operation. On the other hand, reducing the cost and the time to market is mandatory for obvious competitiveness reasons. In this context, the definition of optimized engineering practices is imperative. Automatic code generation has already proven very effective in shortening the time to market and in guaranteeing the quality of the generated code especially if the code generator is qualified. Since 2002, Airbus has extensively used a software development process including Qualified Automatic Code Generation (ACG) with significant and measured savings:

- Airbus has never experienced problems in flight due to coding errors in the Flight Control System software.
- Software development time was divided by 3 (including verification activities), compared to a process without ACG.

In the context of space applications, several experiments were conducted to assess the advantages of automatic code generation. In particular, Astrium demonstrated through several projects and experiments (e.g., DIVAS [1] and LOLA [2]) that automatic code generation brings a real gain in efficiency in the context of fast prototyping for systems requiring complex algorithms and multi-tasking. Astrium also conducted internal studies (e.g., SCAOGEN) to evaluate the use of modelling and automatic code generation in the regular system and software development process of the satellite Attitude and Orbit Control System (AOCS).

The aerospace and automotive industries joined their efforts in order to define a common development workshop for critical embedded systems, using standard formalisms and open-source tools. This objective led to setting up the GENE-AUTO project (www.geneauto.org) in 2006 in the context of the ITEA European research program. Both Airbus and Astrium were members of the GENE-AUTO consortium. The concrete objective of the project was to develop an automatic and qualifiable C code generator from models created with Simulink®/Stateflow® and Scilab/Scicos. The

project ended successfully in December 2008. The GENE-AUTO toolset is currently maintained by its developers and evaluated for industrial usage by several end-users.

The advantages offered by the GENE-AUTO toolset are as follows:

- It was designed and developed based on a common set of user requirements coming from the automotive and aerospace domains and having in mind certification (DO178B level A).
- It is open-source (GPL licence), which favours the long-term maintenance (up to 30-50 years) and the independence with respect to providers' policies.
- Its architecture is modular and flexible. Based on model-driven engineering concepts, GENE-AUTO consists in a set of elementary tools and intermediary models that make it highly evolvable. In particular, the users are able to define their own supported block library (i.e., users can define the code to be generated for model blocks that are not supported by the public GENE-AUTO toolset), or the code optimizations that are required for a specific hardware target. Also, even though the C language has initially been selected as target, it is possible to target a different language, e.g., Ada, with a minimum amount of effort.
- It represents a proof that formal methods can be successfully integrated in such tools. Indeed, a proof assistant was used for implementing one of the GENE-AUTO elementary tools [3], [4].

This paper presents the experiments conducted in 2009 by Airbus and Astrium with the GENE-AUTO code generator, based on models currently used during system design and that are the basis for software development. The remainder of the paper is structured as follows. Section 2 gives an overview of the experimentation context both from the point of view of the technical objectives and of the industrial organization between the toolset users and its developers. Section 3 sketches the characteristics of the selected case studies, while the main experimentation results are summarized in Section 4. Section 5 concludes the paper.

2. Context and Industrial Organization

Both Airbus and Astrium had conducted earlier experimentations during the GENE-AUTO ITEA project. Some of the results have been presented in [5]. The subsequent experimentations that we have conducted in 2009 aimed at improving both the users' confidence in the toolset and the maturity level of the GENE-AUTO technology.

The second objective mentioned above can only be fulfilled if the technology evolves based on the user feedback. This is the reason why we have set up a maintenance contract between Airbus and Astrium as users on the one hand and Alyotech and IB Krates as developers of GENE-AUTO on the other hand. The adopted maintenance scheme had the advantage of federating the efforts both on user and developers sides. From a practical point of view, a common web-based tool was used for the support requests. Periodic Change Control Boards gathering all stakeholders allowed prioritizing the actions with respect to the available budget, distributing the charges and workload and deciding about the contents and dates of new releases.

Most of the work has been performed in the framework of the OBSYS EADS demonstrator, which gathers the various EADS Business Units in their efforts towards engineering process optimization and practical integration of new technologies.

3. Case Studies

3.1. Flight Control Logic & Laws (Airbus application)

The targeted objectives of the Airbus case studies were related to two topics: system design and software design, as follows.

From the system design point of view, the aim was:

- to enhance the current design practices by using modelling features such as automata, vectors and buses,
- to experiment the ability to modify and to mix existing formalisms (data flow named SAO – former version of SCADE) with Simulink®/Stateflow®, taking advantage of the expressiveness power of new modelling concepts without having to completely re-specify existing specifications.

From the software point of view, the aim was:

- to participate in the maturation of the GENE-AUTO toolset by using it on current specifications (within a DO178B software development context),
- to test the ability to customize the source code produced by GENE-AUTO for specific needs (coding standards, optimization, reducing CPU consumption),
- to evaluate the generated code, in terms of CPU and memory consumption, on a real HW target,
- to evaluate the integration of code generated by GENE-AUTO with legacy code.

The code generated by GENE-AUTO was compared with reference code generated by tools currently used at Airbus**.**

The model was developed using the R2006b release of TheMathworks™ tools (Simulink®, Stateflow®).

The performance analyses (CPU time and memory consumption) were carried out by using logical analysers dedicated to the target HW (Intel 486).

In the first case study (named *StickPriority*), a part of the Flight Control Logics was re-designed using state-machines to improve expressiveness of the specification. From a practical point of view, in this first scenario an existing SAO node (pure data flow) was replaced by a Simulink® model calling a state machine. As we can see in Figure 1, this Simulink® model includes a part of the existing data flow. The use of the automata allows identifying directly the five states previously buried in the pure data flow.

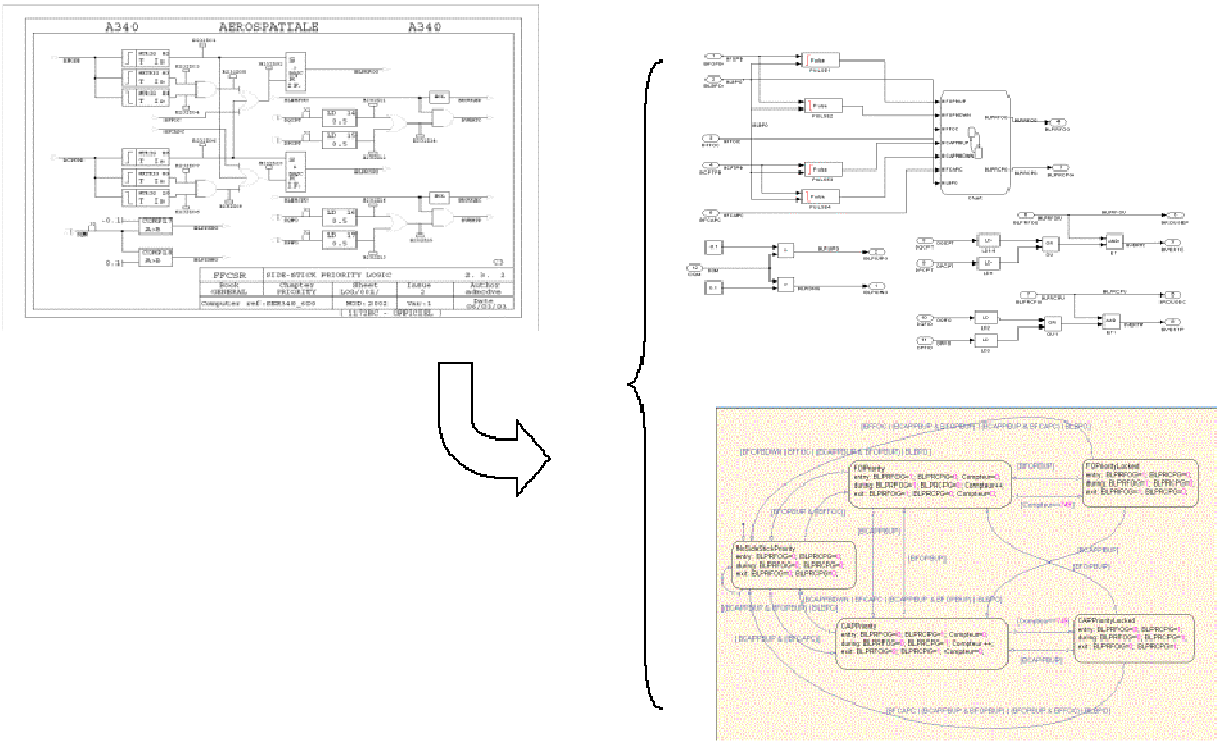

Figure 1: Re-designing the Original Model to Include State Machines

In the second case study (named *MIMO Corrector*), a part of the Flight Control Laws was re-designed using discrete vector based model (see Figure 2). 38 SAO nodes similar to the one shown in Figure 2 are necessary to represent a Kalman filter defined in Simulink® as MIMO correctors.

The interest of the use of vector and matrix computation in the model is obviously the optimization of the size of the specification (38 SAO nodes vs. 2 Simulink® nodes) and its readability.

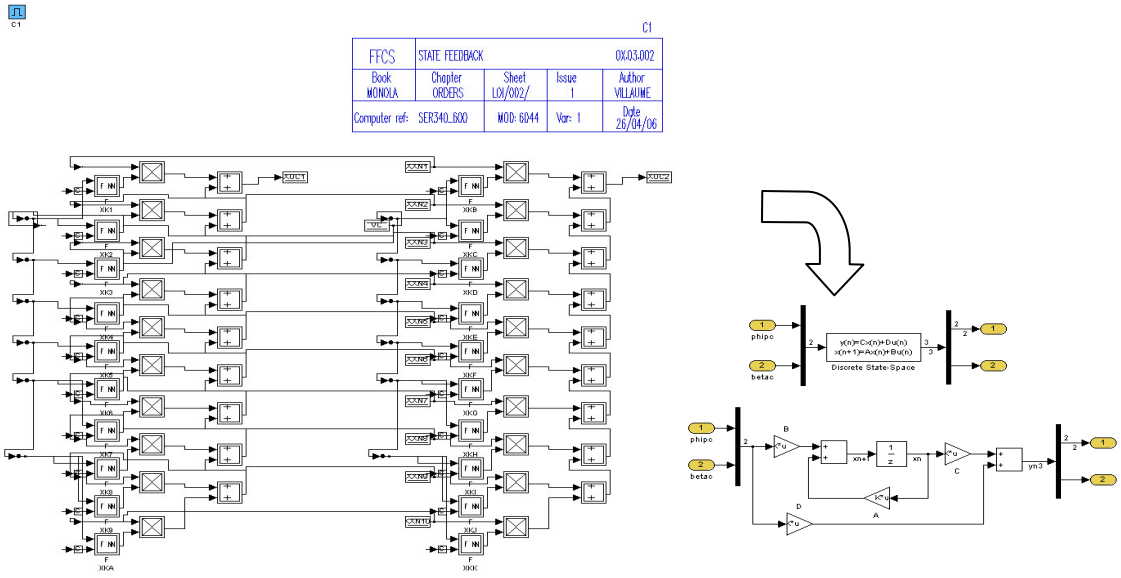

Figure 2: Re-designing the Original Model to Include Vectors

This MIMO corrector is composed of two components: the corrector C1 in Figure 3 (activated at a sequencing cycle of 10ms) and the pre-command C3 in Figure 4 (activated at a sequencing cycle of 40ms). C1 is a purely vector model, while C3 uses a mix of scalar and vector symbols.

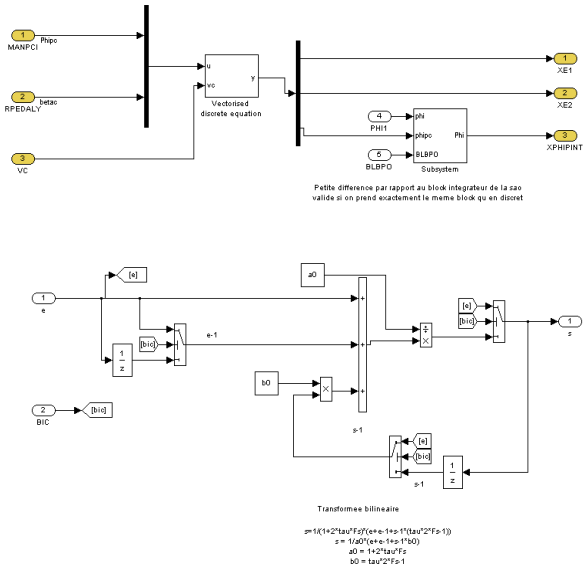

Figure 3: MIMO Corrector C1 Simulink® Model

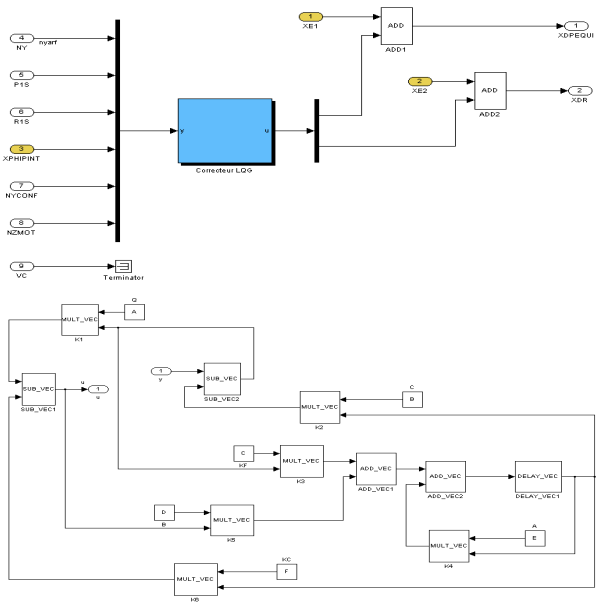

Figure 4: MIMO Pre-command C3 Simulink® Model

Modifications for compliance with the Software context

The data coupling between the Simulink® model and the unmodified parts of the SAO specification induced the necessity to define interfaces (types and names), to have a consistent global specification and to allow GENE-AUTO to propagate types.

Within the Simulink® model all used symbols were defined as subsystems (to have the same interfaces as SAO symbols). Each subsystem was defined with Simulink® native symbols. For each of them an occurrence number was affected within the mask parameter (they are used for code in-lining).

Software production

Two different pieces of software were developed from each model:

1. For the first one (*GeneAuto_Pur*), the C source code was produced by GENE-AUTO with no code generation option.
2. For the second one (*GeneAuto_Bib_Embarque*), GENE-AUTO was used to replace Simulink® symbols by a set of macros call written manually (Airbus macro library used with SAO specification). To allow GENE-AUTO to support the platform specificities (and to allow to re-use of the Airbus library) for the symbols, its standard block library was extended through a set of user-defined typers and backends (the source code for the vectorial symbols was developed from scratch). Relevant symbols were AND, COMP1, LD, OR, PULSE1, PULSE2, ADD, ADD_VEC (vector addition), DELAY, DELAY_VEC (vector delay), DIVR, LIM, MULT, MULT_VEC (matrix / vector multiplication), SUB, SUB_VEC (vector subtraction), SWITCH_N.

For these two pieces of software, the code generated by GENE-AUTO was a reusable C function. Wrapping code was developed for updating the input parameters before the call of the function and the output parameters afterwards.

3.2. AOCS System (Astrium application)

The case study selected by Astrium is based on a recently-developed demonstrator for the new avionics features of the AstroSat 250 platform (targeting low earth orbit missions). The AstroSat 250 platform is based on a new computer including a LEON3 processor, on a new generation start tracker and on a gyroless attitude and orbit control mode running at 16Hz (based on star tracker and control moment gyro actuators). The architecture of the demonstrator is presented in Figure 5.

The on-board software (OBSW) and its environment are both specified in a closed-loop Simulink® model. It is noteworthy that there is strict separation between Simulink® models of the OBSW and of the environment. In the original demonstrator, C code was generated from the model of the application software within the OBSW (the data handling layer was manually coded) and the environment model by using the Real-Time Workshop (RTW®) code generator provided by TheMathworks™. The generated pieces of code corresponding respectively to the OBSW and to the environment have been each installed on different computers connected with the actuators through a network. The goal of the experimentation with GENE-AUTO was to replace

the code generation of the OBSW application software that was originally performed with RTW® by a code generation with GENE-AUTO.

The OBSW Simulink® model is formed of 3186 blocks architected in 10 hierarchical levels. To be able to run, about 2000 constants must be initialized prior to the execution (originally, the initialization was done through the workspace by loading a .mat file). Matlab® functions (e.g., trigonometric functions) are extensively used in the model, as well as Embedded Matlab® blocks containing control laws and algorithms representing mode automata. The model also contains a large number of execution control blocks such as Switch and SwichCase.

Two Stateflow® charts were also added on the one hand to evaluate the approach of coupling dataflow with Stateflow® models and on the other hand to fully test the capabilities of GENE-AUTO.

The model has been developed using the following releases of TheMathworks™ tools:

- Matlab® 7.5 (R2007b)
- Simulink® 7.0 (R2007b)
- Stateflow® 7.0 (R2007b)

Before being processed by GENE-AUTO, the model was saved under Simulink® version R14-SP3 (the version officially supported by GENE-AUTO).

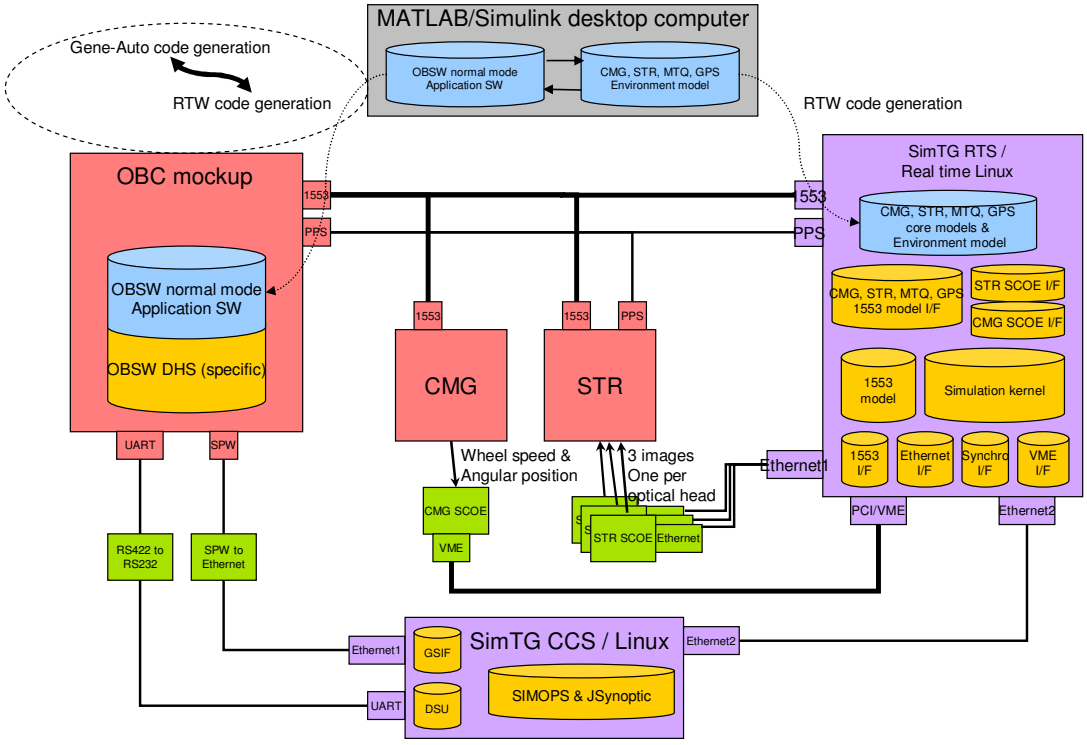

Figure 5 : Demonstrator Architecture

In addition, we have performed several modifications on the legacy model. They fall into the two following categories that are further detailed in the next paragraphs:

- Modifications for compliance of the input model with the GENE-AUTO scope.
- Integration of Stateflow® features in order to evaluate on the one hand the approach of coupling dataflow with Stateflow® models and on the other hand to fully test the capabilities of the GENE-AUTO code generator.

Model Modifications for Compliance with the GENE-AUTO Scope

Parameter Initialization: Usually, the Simulink® simulations require the initialization of a number of parameters before execution. This can be performed in two ways: either by executing a Matlab® script stored in an .m file, or by loading a workspace image (load a binary .mat file). In our legacy model, the initialization is performed by loading a .mat file, while GENE-AUTO is able to parse simple .m files that perform constant initializations. It is not able to read binary .mat files[1]. Thus, we needed to create a .m file from the .mat file. For that, we used a rather artisanal method. We have been able to extract automatically the scalars by using a Matlab® script but this method did not succeed in retrieving the structures. We have thus retrieved them manually by inspecting the workspace once the .mat file has been loaded.

Matlab® Expressions: Currently, GENE-AUTO does not support the use of Matlab® expressions in `Constant` blocks. Our legacy model uses such expressions on a regular basis. Sometimes it is

[1] Such files are built directly in Matlab®/Simulink® by saving the workspace and their structure is not public.

simpler to use such expressions than to use Simulink® blocks. However, they may be replaced by Simulink® blocks (e.g., arithmetic and trigonometric operations) that are currently supported by GENE-AUTO.

Unsupported Blocks: Our legacy model was not created having in mind the blocks supported by GENE-AUTO. A number of unsupported blocks were thus present in the model. Some of them could be replaced by equivalent sets of supported blocks.

Embedded Matlab®: Our legacy model made extensive usage of Embedded Matlab® blocks for representing algorithms and mode automata. Since GENE-AUTO supports legacy code but currently has a limited Matlab® support (only for usage in the initialization .m file), we decided to replace the Embedded Matlab® blocks representing algorithms by S-functions generated with RTW®. This also allowed us to experiment the integration of code generated by different tools. This topic is further detailed in section 4.

Integration of Stateflow® Features

Our legacy model included two mode automata represented by Embedded Matlab® blocks. While the other Embedded Matlab® blocks were replaced by S-functions, we decided to use Stateflow® to model the two mode automata. The Embedded Matlab® representation of the mode automata is a function with data inputs and outputs (the data ports of the block) in which the output values are set depending on the values of the commands received on the input ports (global if-then-else control structure). The translation of such a representation into a Stateflow® chart requires identifying the different states (the different "if" conditions) and the rule allowing to set the outputs depending on the internal state and the values of the inputs. The chart if formed of states, transitions and junctions. Transitions are guarded by conditions referring to inputs and sometimes specify actions represented as output port data updates. The states specify entry and exit actions for updating the output data ports, e.g., the one corresponding to the current mode. Junctions are used as a simplification facility allowing to represent pseudo-states (states with no associated action but necessary to separate more complex control paths). Figure 6 depicts the translation process for the mode automaton from the Embedded Matlab® block to the Stateflow® chart.

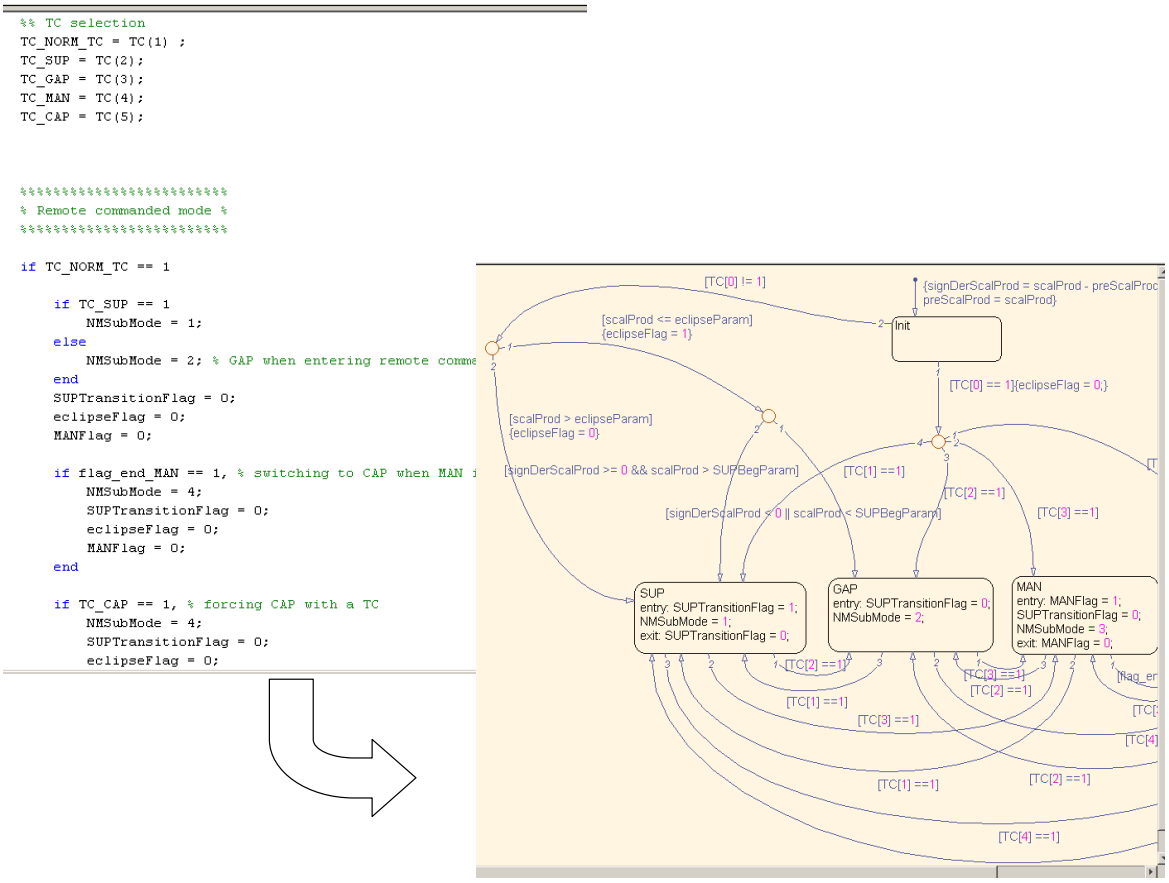

Figure 6: Mode Automaton Translation from Embedded Matlab® to Stateflow®

4. Experimentation Results

Our experimentations aimed at evaluating the generated code from different perspectives (e.g., code correctness, size, quality, traceability between model and code, CPU and memory consumption) and at investigating the straightforwardness of the integrating of generated code with legacy code. Our results with respect to the various topics are summarized in the following paragraphs.

Generated Code Correctness

Astrium read parts of the generated code to asses its correctness. The generated code was executed in the model loop.

Airbus has even performed a functional verification of the generated code on the real target. During this functional verification, the generated code was stimulated by using standard input value sets and we observed identical behaviour on its outputs as the reference software used in a real context.

These activities allowed us to identify few issues regarding the generated code, such as:

- Incorrect use of constants with values between 0 and –1.
- Without optimization option, GENE-AUTO produces a large amount of memory affectations (temporary local data).
- The headers used in the generated code include a “math.h” (which is not qualified in the embedded software context).
- Matrix product did not use the correct index (computation with a value out of array limit).
- Wrong type propagation between double and simple precision for floating point numbers.

The identified issues were reported and most of them are already solved (for the experimentation purpose, the generated code was corrected by hand). We have not found many bugs in the generated code, but on the other hand, we have a number of suggestions for extending GENE-AUTO to support more Matlab®/Simulink® functionality. In particular, Astrium suggested a concrete list of additional Simulink® blocks and a subset of the Embedded Matlab® language.

Code Size

In our previous experimentation Astrium had compared GENE-AUTO generated code with manual code derived from the same specifications and found that the two were comparable in terms of size, number of comment lines and complexity.

In the current experimentation, Astrium compared the GENE-AUTO generated code with RTW® generated code. We noted that the GENE-AUTO generated code size (measured in LOC) is three times smaller than that of the RTW® (RTW® standard edition) generated code from the same Simulink® model. Also, the amount of comment lines for GENE-AUTO generated code is higher (27%) than for the RTW® generated one (16%).

Traceability between Model and Code

Traceability between model and code was analysed by reading the generated code and comparing it with the input model. We have not identified any particular problem (the analysis was conducted on code generated without using the code optimization module of GENE-AUTO). The comments placed in the generated code point back to the original model elements. It is noteworthy that the names given to intermediary variables and input/output structures corresponding to blocks are rather unreadable if the corresponding names are not specified in the input model.

CPU and Memory Consumption

Code generated by tools currently used at Airbus (named *reference_SAO*) was used as reference for the comparison of CPU and memory consumption.

The generated software releases did not reach the expected performance level for several reasons (non-optimized legacy code for vector computation, need for updating of the input/output parameters of the C function, extensive uses of temporary local data, use of double precision in the input model).

The exploitation of the code optimizer of GENE-AUTO (which suppresses intermediary useless variables) and the use of simple precision variables (instead of double) allowed obtaining results similar to those of the reference code in terms of CPU performance.

We observed an increase of the memory consumption (especially the RAM) when using vectors with GENE-AUTO. This is once again due to the important use of local variables instead of global ones (as it is done in the reference code).

The ROM consumption is also greater, due to the use of more complex computation (matrix computations, for loops, etc.).

Code Quality

During the ITEA European project, we had used commercial tools Logiscope™ Rulechecker in order to evaluate the generated code quality with respect to the Astrium C-coding standard that is used for manual code. The additional experimentation conducted in 2009 did not reveal any changes in the characteristics of the generated code: most of the coding rules are fulfilled. Only a naming rule is violated very often but the names in the code depend mainly on the existing names in the input model, which should follow the naming rules set for the manually-written code. MISRA rule 11 is also often violated (Identifiers shall not rely on the significance of more than 31 characters). Longer variable names do exist in the code. However, these names are derived from the model.

Integration of Generated Code with Legacy Code

In a real operational environment, we can expect to have to integrate legacy code (manually written or generated with another tool) with the generated code. Therefore, it is interesting to investigate the ways integration may be performed.

The Astrium case study model includes many Embedded Matlab® blocks, which are currently not supported by GENE-AUTO. As a consequence, our strategy was to generate code with RTW® from the Embedded Matlab® blocks and then integrate the code generated with RTW® and GENE-AUTO.

There are several ways to integrate parts of code generated by GENE-AUTO with parts of code generated by another tool from a given model. The first method is manual and consists in the following steps:

- Masking all subsystems that are not to be generated by GENE-AUTO thus forcing the code generator to ignore them (a comment will appear in the generated code as replacement of the corresponding function call).
- Generating the code from those subsystems with another code generator.
- Integrating the pieces of code by hand.

Another method consists in the following steps:

- Generating the code from the selected subsystems with another code generator than GENE-AUTO.
- Replacing those subsystems inside the model by legacy code S-functions (using the generated code).
- Generating code from the whole model with GENE-AUTO.

We have selected this second method, since part of the integration work is performed automatically by the GENE-AUTO code generator. Indeed, GENE-AUTO is able to parse legacy code S-function blocks and call the associated legacy code from the GENE-AUTO generated code, provided that the corresponding S-function blocks are masked and that the mask properties are set properly.

In practice, the integration of the RTW-generated code back into the model requires using a wrapper, as the RTW-generated function has no input parameters and no return values (the inputs and outputs of the block are stored in a globally declared structure containing one data for each port). On the other hand, the convention for a legacy code S-function is that each return value is converted into an output port of the corresponding Simulink® block and each input parameter is converted into an input port of the block. Hence, it is necessary to wrap the RTW-generated S-function in order to import it back again in the Simulink® model with explicit input and output ports. The wrapper function assigns the values of the parameters of the wrapper function to the elements of the globally declared input structure. Then it calls the automatically-generated function. At the end, it assigns the values of the globally declared output structure elements to the output parameters of the wrapper function.

Software Investigation

Airbus conducted experiments for the introduction of “investigation variables” or probes represented in the Simulink® model by scopes to give visibility to some internal variables. A particular backend was developed to exploit the scopes in the generated code. These extra-variables used here for the software debugging phase would be also very useful for the validation teams. This has to be further studied in order to determine the relevant (necessary and sufficient) states or signals we need in order to perform the system validation.

5. Conclusion

The experimentation conducted in the context of the OBSYS EADS demonstrator met its objectives, i.e., increase both our confidence in GENE-AUTO and the maturity level of the toolset.

We demonstrated the functional equivalence between the reference code/model and the code generated by GENE-AUTO from models using different formalisms (Simulink®, Stateflow®, SAO). The use of automata and vectors improve the readability and ease the understanding, thus the maintainability of a system specification.

We found few bugs in the current GENE-AUTO toolset, whose maturity level is rather good, taken into account its age and history. On the other hand, we have identified useful enhancements for the toolset. In particular, we believe that enriching the supported block library would be of much help for potential users. We have suggested a list of additional blocks to be supported. Moreover, taken into account the confirmed trend to replace C code S-functions by Embedded Matlab® blocks in Astrium Satellites, support for Embedded Matlab® becomes very important. More generally, further investigations related to the code generation could focus on the use of local variables vs. global variables and optimizations (e.g., temporary local data, matrix computation to avoid for example full computation when matrices are not full).

It was demonstrated that GENE-AUTO allows to customize and to optimize the source code, similarly to the Airbus code generator. The results in terms of performance (with the right options and with the last releases of the toolset) are similar to those of the Airbus code generator.

This experimentation allowed us to estimate the maturity of the technology of the GENE-AUTO code generator, which we consider having reached a level of 5 for Airbus and 4 for Astrium (Airbus conducted

experimentation on a real case study involving operational teams with system designers and software developers, while Astrium conducted its experimentation on a representative demonstrator case study). Improvement of performance and extension of capabilities of the generator are still necessary before bringing the tool into the industrialization phase.

We found that the maintenance organization that we have set up was effective, allowing to federate efforts both on user and provider sides. The OPEES initiative should enforce these conclusions and the long-term maintenance of the toolset through a community of users and developers.

6. Acknowledgements

The authors would like to thank the toolset development and maintenance team from IB Krates and Alyotech for its valuable support and responsiveness.

7. References

[1] P. Diris, et al.: “Astrosat 250 Core Avionics Hardware in the loop Demonstration using Fast prototyping Technologies”, Data Systems in Aerospace (DASIA 2009), Istanbul, Turkey, 2009.

[2] http://www.astrium.eads.net/en/prog/lola.html

[3] N. Izerrouken, et al.: “Certifying an Automated Code Generator Using Formal Tools: Preliminary Experiments in the GENE-AUTO Project” in ERTS’08, Toulouse, France, January 2008

[4] N. Izerrouchen, M. Pantel and X. Thirioux: “*Machine-Checked Sequencer for Critical Embedded Code Generator*”, Springer LNCS proceedings of the International Conference on Formal Engineering Methods (ICFEM'09), pp. 521-540, Rio De Janeiro, Brazil, December 09.-12. 2009.

[5] A.-E. Rugina, et al.: "*GENE-AUTO: Automatic Software Code Generation for Real-Time Embedded Systems*", Data Systems in Aerospace (DASIA 2008), Palma de Majorca, Spain, 2008.

8. Glossary

ACG: Automatic Code Generation

AOCS: Attitude and Orbit Control System

LOC: Lines Of Code

MIMO: Multiple Input Multiple Output

OBSW: On-Board Software

RTW: Real-Time Workshop

SAO: Specification Assistée par Ordinateur (Computer-assisted specification)