New methodology to develop certified safe and secure aeronautical software - An embedded router case study

Antoine Varet, Nicolas Larrieu

NEW METHODOLOGY TO DEVELOP CERTIFIED SAFE AND SECURE AERONAUTICAL SOFTWARE – AN EMBEDDED ROUTER CASE STUDY

Antoine Varet, Ecole Nationale de l'Aviation Civile (ENAC), Toulouse, France
Nicolas Larrieu, Ecole Nationale de l'Aviation Civile (ENAC), Toulouse, France

Abstract

New aeronautical traffic profiles are growing in usage and complexity. Higher throughputs and new opportunities could be served by multiplexing some different data but the heterogeneity of their safety and security constraints remains the main problem for promoting multiplexing solutions through a unique network link. For this purpose we are producing an IP based Secure Next Generation Router (SNG Router). This SNG Router provides regulation, routing, secure merging of different data sources as well as preserving their segregation.

In order to ease the SNG router development we defined a new methodology for the process of aeronautical software development. This methodology permits us to rapidly transform verifiable models into a safe and secure byte-code certifiable at DO-178B highest levels with reduced costs. This paper presents the methodology tool chain, which uses a qualified model transformer to generate code for a secure virtualization infrastructure with controlled inter-partition communications. A separation kernel running on an embedded target enforces the segregation of computations done on the data. The case study of the SNG Router illustrates concretely how the methodology can be conducted.

Introduction

Heterogeneous aeronautical networks

Future aeronautical environment is in continuous evolution and new traffic profiles are growing in usage and complexity: new services are added to the Aeronautical Traffic Communications (ATC) and its two subclasses which are Aircraft Communication Domain (ACD) for the flight management system communications and Airlines Information Services Domain (AISD) for the flight helper system communications. Airline communication needs are growing with new usages of the In-Flight Entertainment networks (IFE) and facilities for passengers could appear in a near future in the context of the Aircraft Passengers Communications (APC).

These new opportunities need higher throughputs and more security such as confidentiality, integrity, availability and authentication. Multiplexing different communications through a unique network link, such as the ATC with the APC on a single Satellite Communication Data link, could serve these requirements but the heterogeneity of the communications and their independence remain the main problem for promoting multiplexing solutions.

In order to overcome these issues, we are elaborating an IP based Secure Next Generation embedded Router (SNG Router) in collaboration with the firm Thales Avionics France, providing regulation, routing, secure merging of different data sources and preserving their segregation.

During the design process we were searching for a set of solutions to minimize the certification, design and development costs and maintain a high level of security and therefore we have built the development methodology we present in this paper.

The structure of this document is as follows: first we will introduce some safety and security standards used in the aeronautical world, and then we will present some tools we need to instantiate the methodology. In the second section, the methodology is explained in detail then a specific tool chain is developed to instantiate the different steps of the methodology. The last section of this paper illustrates how we can apply our methodology with the design of our embedded SNG Router case study.

Safe development for aeronautical software

Any critical embedded system in avionic suite must be certified against dysfunctions and their potentially catastrophic consequences. Since 1992

the DO-178B [1] "Software Considerations in Airborne Systems and Equipment Certification" published by RTCA is the aeronautical standard for safety certification of embedded software.

This standard enables a software to be evaluated with different levels of assurance called Design Assurance Level (DAL) between A and E: the lowest DAL-E is required for software which failure has no impact on safety, aircraft operation or crew workload whereas the highest DAL-A is required when a software failure may cause a crash or cause a loss of a critical function. DAL-A-certified software gives more confidence in its safety than DAL-B software but is more expensive in evaluation effort, as described in table 1.

Table 1 DO-178B DAL evaluation costs

Software level	Objectives to satisfy	…to satisfy with independence[1]
DAL-A (Catastrophic)	66	25
DAL-B (Hazardous)	65	14
DAL-C (Major)	57	2
DAL-D (Minor)	28	2
DAL-E (No effect)	0	0

Conforming to DO-178B standard, the load for certification of the final product can be reduced with the qualification of some development and verification tools by enforcing certain properties related to the results of these tools: embedded software can be certified for a specific level of safety assurance for a specific avionic suite, tools to build or check software can be qualified. For instance, a DAL-B qualified compiler can be used to produce a certified binary image of code at level B, C, D (DAL-E does not require certification).

Secure software development: the Common Criteria

Since the beginning of the aviation, safety considerations have been a central requirement in the aeronautical world: verification processes and system redundancies reduce unattended dysfunctions. Avionic systems being in closed perimeters, the security were mainly enforced by physical security for the different areas, reducing the vulnerabilities for potential attackers. Since the September 11, 2001, the place for security considerations has grown: avionic systems are more and more open and securing the different physical perimeters is not sufficient anymore.

Some standards exist to increase the assurance in security of generic software production processes. The ISO/CEI 15408 [2] called "Common Criteria for Information Technology Security Evaluation" (CC ITSEC, or smaller CC) is a framework to specify security requirements and to provide assurance in computer systems.

Like the DAL defined in DO178B, the CC enables the product to be evaluated at a level of assurance, from the lowest Evaluation Assurance Level 1 (EAL-1) to the highest level EAL-7. We can note that formal method usages are mandatory for EAL-5 evaluation and higher.

The application of the CC in our work is in progress (this framework has been indeed used to evaluate some tools presented in this paper) but is beyond the scope of this paper. Some publications deal with the application of the CC requirements in aeronautical processes [3].

A methodology to develop our software

The aeronautical context constrained by safety and security requirements conducted us to elaborate a new methodology to produce rapidly aeronautical software with certification and evaluation effort reduction. The methodology we propose is based on model-into-code transformers and virtualization infrastructures with separation micro-kernel: the qualified transformers generate code and binaries with assurance for safety and the separation micro-kernel enforces MILS principles (Multiple Independent Levels of Security is defined below) which increases the assurance in security. We can note that the conception is the most important for the software safety and security, the tool qualification and the separation kernel usage are additional increases.

Model Checking (MC) and Formal Methods (FM) can be used to enforce the confidence of the

[1] "[…] Independence is achieved when the verification activity is performed by a person(s) other than the developer of the item being verified, […]", DO-178B P.82

safety of the originating models, nevertheless the application of these techniques are out of the scope of this paper.

The separation micro-kernel provides secure inter-partitions communication facilities (S-IPC) which may be compatible with the ARINC 653 [4] Application Executive (APEX) standard. This document specifies indeed a list of services to provide for partitions through API calls: partition and process management, error handling, time management, intra- and inter-partitions communications.

Some qualified code-based model generators

The generation of lines of code from graphical or textual models is the main objective of code-based-model transformers. These programs take models produced by design tools and convert them into code by applying deterministic rules. In this section we will present only few tools useful for illustrating our methodology.

Simulink(R) software [5] is a well-known graphical user interface (GUI) to model, simulate and analyze dynamic systems. This commercial Matlab toolbox enables the designer to model a prototype and test its behavior. The graphical approach of Simulink consists in inserting and linking into a model or its sub-models some blocks representing linear or non-linear subsystems, functions, inputs, outputs, switches... The simulation capacities can support simultaneously continuous and discrete time and provides facilities to define complex inputs and friendly scopes to represent numerical outputs.

Scicos [6] is an alternative for Simulink, the main difference between Scicos and Simulink concerns the licenses: Scicos is free and open source while Simulink is commercial; moreover Scicos is standalone software whereas Simulink needs Matlab.

The Stateflow(R) product [7] completes Simulink by permitting the modeling and the simulation of reactive systems also called event-driven systems, where the finite-state machines can change from a state to another through conditional (event or logic condition) or unconditional transitions. The state models made with Stateflow are subsystems integrated into Simulink system models and called periodically or on request of another subsystem. Stateflow enables to model machines with parallel states (different states active simultaneously), with graphical functions based on flow diagrams, with the temporal logic to increase event scheduling... For our knowledge, it does not exist any alternative to Stateflow to complete Scicos yet.

The files generated by modeling tools can be converted into source code files: these model-to-code transformers may be viewed as compilers of a high level of abstraction. The most known tool in the aeronautical domain to convert models into source code is probably the SCADE development environment [8], commercialized by Esterel Technologies: supporting up to DO-178B DAL-A qualification to produce C and Ada source code, it has graphical design, model test coverage and formal proof verifier capabilities. For instance, Airbus and Eurocopter used it to develop, respectively, the Primary Flight Control Systems for A380 and the Automatic Pilots of EC-135, two highly critical software projects.

Since 2006, the Gene-Auto ITEA European project [9,10,11] aims at building a free and open-source qualified C code generator from systems models. The tool is now in a stable version and can transform Stateflow/Simulink models or/and Scicos models into C library source code; it will soon support Ada language as target. Like SCADE, Gene-Auto has been designed to be qualified for aeronautical safe software development. Free and open-source provides some advantages presented later in section “Different tool chain proposals for the methodology”. We chose to use this tool to develop our SNG Router case study.

Some virtualization infrastructures

A virtualization infrastructure permits to run on a single processor different software components such as applications and operating systems (OS) into “virtual machines” or “partitions”, with isolation between them for some resources or behaviors: the space partitioning designates isolation between the memory spaces used by different partitions while the time partitioning means a scheduling with no interference between the processor time-slots assigned for the different partitions (a partition cannot “steal” the time-slots

assigned to other partitions and then “freeze” the system indefinitely).

Most of time, a “real” OS called “host OS” runs the different partitions, the “virtual” OS running in virtual machines are then called “guest OS”. The host and guest distinction is useful to separate what is outside and inside the virtual machines. The resources (processor, memory, input/output...) used by the guests are controlled by the host. Using different virtual machines on a single real machine may appear inefficient (system complexity increasing) but in fact it can provide some optimizations for load repartitions and a great stability for servers in case of some virtual machine failure by isolating the faults. Figure 1 illustrates how a partition can stop without altering the execution of the other partitions: the unprivileged p3 process raises a fault (to do so, we used the following sample of code *{ int*n=NULL; (*n)++; }* to lead to a segmentation fault), conducting the B partition to halt and then the p4 process to stop its execution; the A partition and its 2 processes p1 and p2 are not affected by the B partition failure.

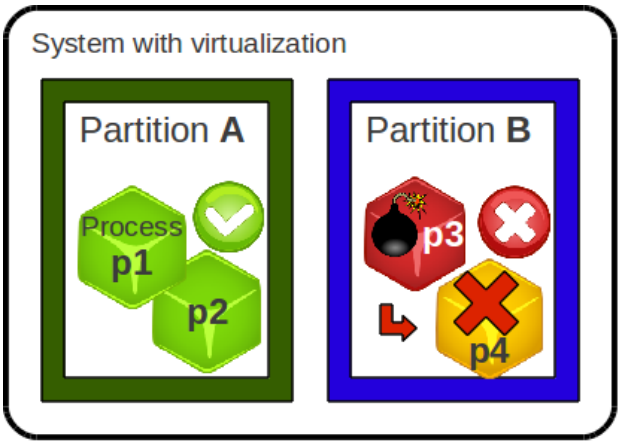

Figure 1 Fault protection enforced by a partition

One technique of virtualization is called the paravirtualization or embedded virtualization [12,13]: during its execution in its partition, the guest software components accesses to the resources through an interface provided by the host. This technique enables guest applications to run faster than with other techniques such as hardware emulation, but needs an adaptation of the guest application to use the resources not directly but through specific host functionalities.

When the host software is dedicated to the paravirtualization, this software is called separation kernel (or micro-kernel). It may provide some Inter-Partition Communications (IPC), some assurances in the scheduling (for real time requirements), in the space partitioning (with the help of some Memory Management Unit)…

WindRiver(R) VxWorks platforms are the current leaders for Real Time Operating Systems (RTOS). Commercialized by the Californian WindRiver Company, the VxWorks product [14] exists in different sub-series: VxWorks (generic version), VxWorks 653 (RTOS advised for DAL-A qualification), VxWorks MILS (this secure RTOS is evaluated at CC31-EAL6+ and then advised for security evaluation of suites based on this RTOS).

Sysgo PikeOS [15] is a concurrent RTOS produced by a Franco-German company. Compliant with the MILS principles and evaluated at CC31-EAL3+ (EAL5+ is in progress), this RTOS can be certified for DAL-A software. APEX and POSIX compliance API for Sysgo PikeOS are optional. It provides the CODEO plug-in for Eclipse IDE which shorts the development phase for Sysgo PikeOS based applications.

Proposed methodology

General presentation

This methodology has been elaborated to develop rapidly safe and secure software. It is based on model transformers and virtualization infrastructures. A separation micro-kernel enables the hardware to run concurrently different applications (software which implements the function) or operating systems (software which provides facilities to run software). Each concurrent execution is associated to a dedicated partition, with specific time and space resources separated from other partition's resources. For instance, a partition has access to a specific area of the memory and cannot alter the memory area assigned to another partition.

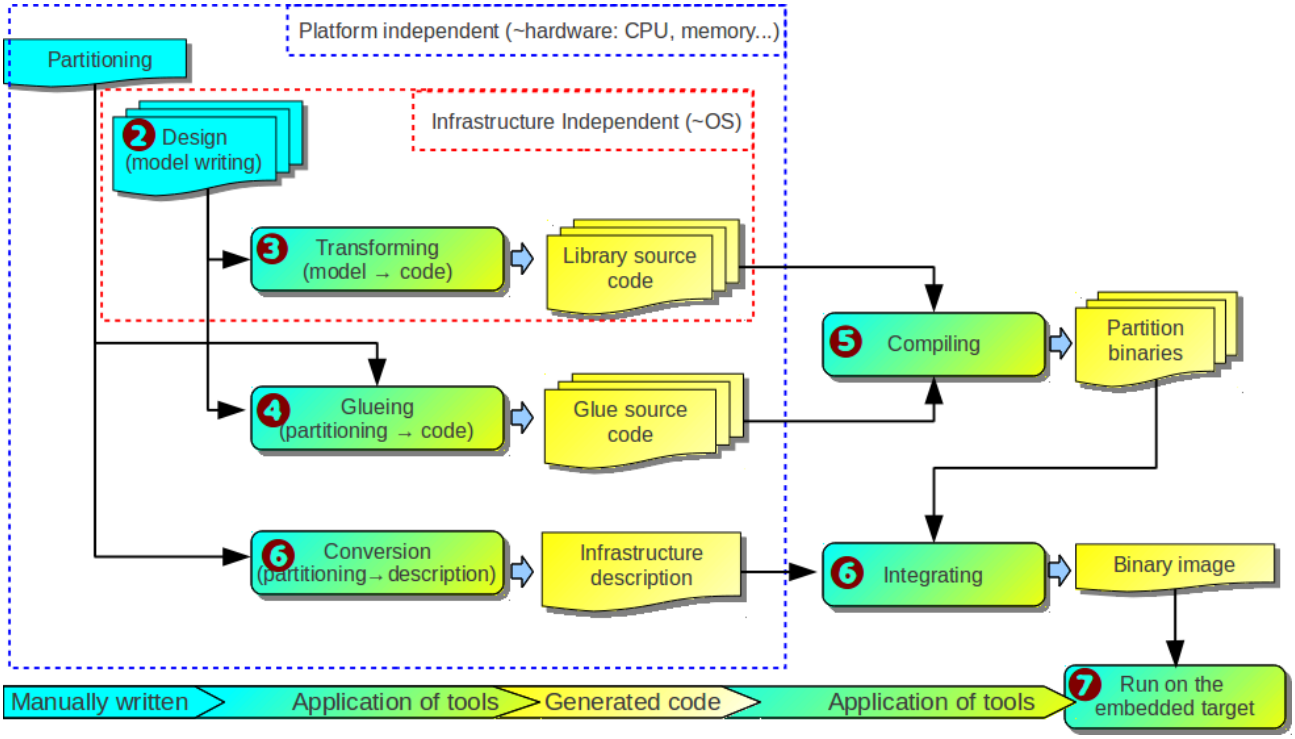

Figure 2 Steps of our methodology

In our methodology, we use the inter-partition communication facilities to enable the different parts of our system to run together. The methodology requires a secure separation micro-kernel to enforce some assurances about the security of the allowed IPC: pipes and shared memory are constrained for data type, length and access. The granularity of the controls is at the partition level. The architect of the system defines all resource assignments and the different IPC for each partition and specifies what is allowed and what is forbidden (more precisely, all operations not explicitly allowed are forbidden for safety and security considerations).

The first step of the methodology presented in figure 2 is the partitioning: the architect divides the future software product into different “partition classes”, each partition class being assigned to a subset of the product functionalities. The product will run a set of instances of these partition classes. The partition instances may communicate with each others. Then when the design of a feature is divided into 2 partition classes, the feature is realized in fact by at least 2 partition instances and all the communications between them.

In a second step, the designer models one implementation for each partition class. A model may be directly implemented in C source code but graphical tools increase the level of abstraction and permit to ease the design, to check more efficiently certain aspects of the models (such as the code coverage) or to ensure the model to have some properties (for instance to avoid any dead code). Figure 3 illustrates the complementary goals of the designer and the architect.

The third step consists of transforming each partition class model into a library source code: the generated code cannot be compiled and run directly, because contrary to application source code, library code has not a single entry point but has one entry point for each exported function and data.

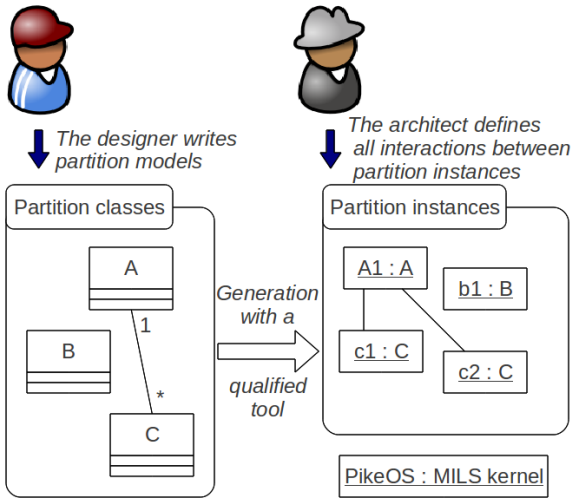

Figure 3 Architect's and designer's tasks

In the fourth step the glue code between the library source code and the separation kernel communication interface is manually written or automatically generated with a tool we will call "the glue maker" below in this paper: this code provides a unique entry point for the partition class; it reads all the inputs of the partition through the S-IPC API, calls the library functions, writes the outputs and redo these three steps indefinitely.

During the fifth and sixth steps the developer compiles codes for all partition classes then assembles and links them together into a binary image containing the separation micro-kernel part. The user-defined architecture from the first step drives the generation.

The developer can now download then run this final binary image on the embedded target or an emulator in order to confirm the correct behavior of the software and its performances.

The steps may be summed up as:

1. Partitioning
2. Design
3. Transforming
4. Glueing
5. Compiling
6. Integrating
7. Running

Advantages

This methodology shorts the process of software development in different ways. High-level models enable designers to verify and correct the behavior of the system as soon as possible in the development phase, reducing the risk of divergences in the final product and then the costs of repair/containment.

The functional decomposition into partitions not only eases the design phase but permit to isolate some underspecified or incorrect functionality and to insert partitions dedicated for safety and security controls over the inputs and outputs of other partitions, easing the instrumentation of the final software. Moreover, the engineer may reuse some partition classes for other projects.

The automatic production of most part of the code reduces some problems of repetitive low-level tasks and bugs generated by the copy-and-paste: for instance, programmers do not have anymore to manage manually some low-level tasks such as shared memory access control or calculation validation because the high-level model semantic and the different transformers perform together these tasks. For example, a division by zero conducts the processor to raise a potentially fatal error and must absolutely be avoided; the transformer can produce a code with a systematic verification circumventing this failure. Moreover, the introduction of hidden channels by a malicious programmer is more complex because there is a smaller quantity of source code and graphical charts to hide them. Nevertheless, a conception error in the transformer may produce a malformed code with an error repeated a lot of times: that is why the transformer must have passed the qualification phase to produce code with a high-level of assurance.

The library code generated from the different models is independent of the virtualization architecture; for that reason the code is easily portable for most systems. In addition, with little lines of glue code it can be smartly tested on any software and hardware architecture, for example on a single laptop with a Linux operating system and gcc compiler, to prove some concepts or to integrate it into other projects without safety constraints, for instance to develop a flight equipment demonstrator.

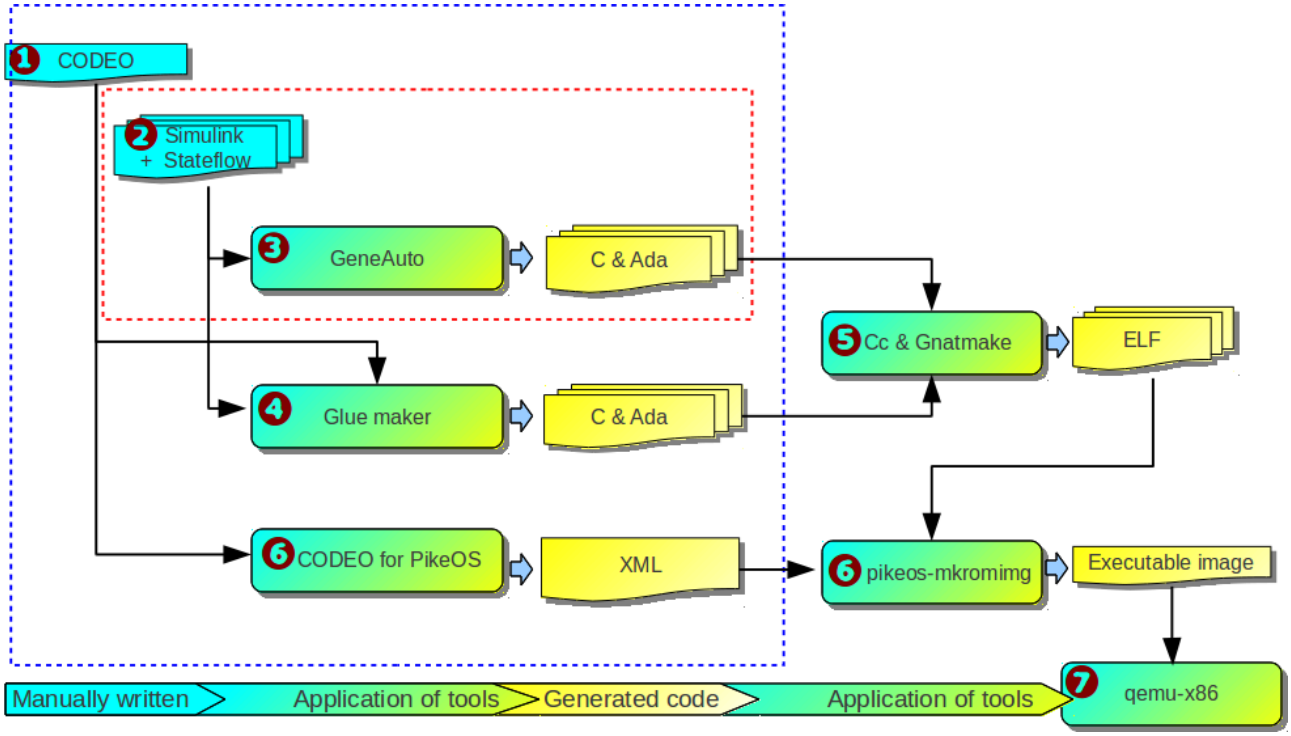

Figure 4 Tool chain "R" used for SNG Router

Other source codes already written (and certified) from predecessor software (in C or other languages) can be reused in the projects developed with the methodology. These codes can be integrated into one (or some) dedicated partition(s) which will communicate with other "legacy" partitions developed from the models.

The secure virtualization infrastructure enforces the segregation between partitions and the control of all intercommunications, involving the safety and the security of the execution of the whole system and reducing the vulnerabilities.

A last but not least advantage of the methodology is the "strong" resource partitioning: S-IPC capabilities are enough restricted to plan to replace an entire software partition by some dedicated hardware solution, reducing CPU consumption and then the constraints related to time scheduling.

Methodology instantiation with existing tools

The previous section presented the main lines of the methodology. Next table 2 resumes a set of tools applicable during the different steps of the methodology.

Table 2 Available tools for the methodology

Step	Usage	Tools
2	Modeling/ Design	Simulink only (graphical) Simulink+Stateflow (graphical) Scicos (graphical) B (textual) [16]
3	Transform	Gene-Auto SCADE ComenC [17]
5	Compile	Gnatmake (Ada) Cc (C) Ocaml (CaML) [18]
6	Separation kernel	Sysgo PikeOS WindRiver VxWorks 653 WindRiver VxWorks MILS
7	Target	VirtualBox [19] qemu-x86 [20] (any embedded product)

The partitioning step (1) must be done manually by the development team, but most separation kernels are sold with an integrated development environment (IDE) easing the integrating step (6). The modeler tools available for the design step (2) and presented in table 2 are detailed in section "Some qualified

code-based-model generators". The transformers of this table enable the developers to pass the transforming step (3) and are explained in the same section. Until now, no qualified tool exists to automate the glueing step (4) because of the dependence on the transformer and on the separation kernel, so the glue code must be written and checked manually. The different compilers of the table 2 support the compiling step (5). The running step (7) may be done on a real embeddable target or any virtualization tool able to emulate different targets.

Before applying the methodology, the developers must choose a specific subset of the tools presented in the table 2; this subset will be called a "tool chain" in the rest of this paper, because the tools have to be applied the one after the others. For instance, to develop our router case study, we have chosen to employ a tool chain named "tool chain R" ("R" like the first letter of Router) and presented in figure 4. In such a chain, models are drawn with Simulink and Stateflow graphical modelers, the transformer used is GeneAuto, the compiler is a qualified C compiler "cc", the separation kernel is PikeOS and the development target is qemu-x86[2]. The PikeOS micro-kernel is provided with its IDE "CODEO" [21] (a plug-in for Eclipse) which facilitates partitioning (1), compiling (5), integration (6) and running (7) steps.

With our tool chain, there is no qualified glue code generator to link the GeneAuto generated code with the PikeOS separation kernel. So the glue code may be written manually but in fact we developed small unqualified programs to automate this step (for certification requirements, the tool must be qualified or the resulting source code must be checked by developers with usual methods such as code auditing). Two usages are possible for the glue code. In "regular" usage, the glue code algorithm is the easy "read-compute-write" algorithm: all inputs of the partition class are read, then the generated library function is called with the appropriate parameters, then all outputs of the partition are written, and these three actions are repeated in an infinite loop. In addition, the glue code enables the model to exploit the "triggered" usage: the model can invoke explicitly the input reading or the output writing functions at any time, for example to refresh data or to emit several output messages for one incoming message. Figure 5 illustrates the algorithm of a partition where the regular mode is completed by a triggered call to output something. The regular usage is mandatory for all partition classes whereas the triggered usage is optional, for specific requirements.

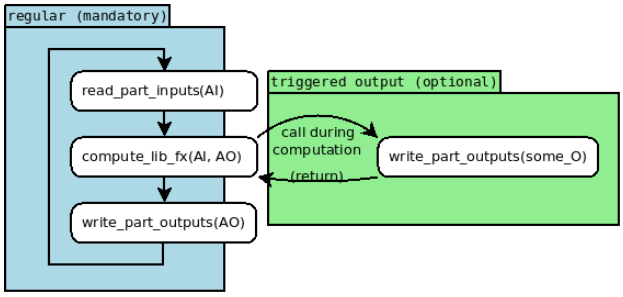

Figure 5 Glue code regular and triggered usages

Advantages of this specific tool chain

This specific tool chain "R" has the following advantages. For the safety evaluation process, usage of a qualified transformer (Gene-Auto) plus a qualified compiler (cc) reduces the load of certification: the source code is automatically developed; Gene-Auto ensures the code to comply with low-level requirements and software architecture, to be verifiable, conform to standards, traceable to low-level requirements, accurate and consistent. Moreover, the qualification of the compiler implies the executable object code to comply and to be robust with low-level requirements. Some test activities are reduced: the qualification ensures that test procedures and test results are correct. They cover all software requirements, and cover the entire code structure while easing the explanation of discrepancies. To summarize, the benefits affect 16 over the 66 evaluation tasks required for DO-178B DAL-A certification! They are resumed in the table 3 below. Thus, the time and money costs of the qualification of the tools are probably a small overload largely counterbalanced by the gain on the costs of classical certification tasks.

[2] Qemu-x86 is an emulator of a Intel x86 processor compatible hardware

Table 3 Tool chain R contribution to certification

Task	Description
A.2.6	The source code is developed
A.2.7	The Executable Object Code is produced
A.5.1	The source code complies with low-level req.
A.5.2	…complies with software architecture
A.5.3	…is verifiable
A.5.4	…conforms to standards
A.5.5	…is traceable to low-level req.
A.5.6	…is accurate and consistent
A.6.3	Executable Object Code complies with low-level req.
A.6.4	…is robust with low-level req.
A.7.1	Test procedures are correct (partly)
A.7.2	Test results are correct, discrepancies explained (partly)
A.7.4	Test coverage of low-level req. is achieved
A.7.5-7	…of software structure is achieved

Whereas the methodology does not explicitly require any security evaluation, usage of an evaluated MILS separation micro-kernel provides some assurances to ease the evaluation of the whole product: PikeOS is already evaluated at the level EAL3+ of the CC (and would be soon at EAL5+). This separation kernel is an important step to build an entire secure product and to evaluate it easier than classical monolithic software. Nonetheless a system cannot be completely secure with just one item; security must be enforced by a set of requirements applied at different steps of the development process. Indeed a separation micro-kernel is not sufficient to ensure the security of the whole system.

Different tool chain proposals for the methodology

The adaptability of the methodology with different tool chains enables the engineer to diversify versions of the product for a better safety assurance. For instance, a producer may use concurrently the two next tool chains to develop its product:

- Tool chain C (Commercial): the graphical models are designed with Simulink(r) and converted into C source code by SCADE software suite, and then the cc compiler generates the binary to run in VxWorks(r) 653 operating system.
- Tool chain F (Free): graphical models are designed with Scicos and transformed by GeneAuto into Ada source code, then the gnatmake compiler produce the binary code to run on PikeOS[3](r).

Tool chain C and tool chain F are clearly dissimilar, but both are "similar" to develop: the methodology imposes such chaining of tools and both tool chains have the same safety evaluation benefits.

However, the tool chain F has the following advantages: almost all tools are free and open-source. This tool chain eases the appropriation of the code for new programmers in existing projects, it improves the deployment for new customers (they can find themselves the responses to their questions without spending time for asking and waiting support), it improves the adaptability (correction of small deviations needs less time), it increases the re-usability of the code and the long-term maintainability (source codes of the tools are open-source, then the users can copy and archive them or branch them to extend the tools), it reduces the cost for buying and managing the tools licenses and avoid the cases of developer unable to work because there is no token available anymore for the development software use.

Illustration of the methodology: the SNG Router

Product description

Since September 2010, the French Civil Aviation University (ENAC) has conducted a thesis in cooperation with industrial Thales Avionics Toulouse in the field of future aeronautical networks, especially in direction of safety and security considerations. The proof of concept of a safe Secure Next Generation IP Router completes the thesis. Objectives of this product are firstly the routing of Ipv4 and Ipv6 packets between different networks with different levels of criticality and secondly the

[3] PikeOS is not free. All other tools of the tool chain F are free.

multiplexing and the securing of the packet transmissions into shared links. For instance, a single SatCom data link may transmit both ATC and APC data. For our study case, we will produce a router with a classical Ethernet interface, a secure one and an administration one. Figure 6 presents such a configuration.

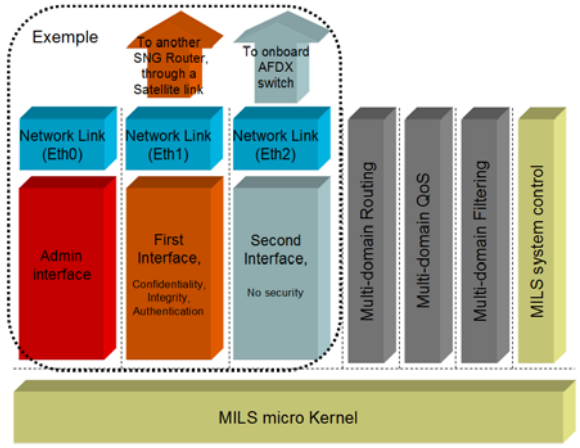

Figure 6 A SNG Router example configuration

To develop our SNG Router, we decided to use the tool chain R: Simulink and Stateflow are the modelers, GeneAuto is the transformer, cc is the C compiler, Sysgo PikeOS is the separation micro-kernel and qemu-x86 emulates the target. In this section, we will apply step by step the methodology in order to develop the embedded software and byte-code of the SNG Router.

Partitioning stage

The first step of the methodology consists to divide into parts the functionalities we want for the product: the SNG Router will run several partitions for routing and filtering IPv4 and IPv6 packets plus some partitions to interface the hardware network interfaces with the software routing and filtering partitions. Moreover, some network links need secure communications, so we introduce some security enforcement partitions. In addition, we will need an additional partition for the administration of the router, the statistics management and some miscellaneous tasks.

Therefore, the decomposition of the functionalities presented in figure 7 gives:

- a partition class Piface to interface the software with the hardware network device,
- a partition class Pfr, declined in two subclasses, Pfr4 for filtering and routing IPv4 packets and Pfr6 for IPv6 packets,
- a partition class Pse to secure some dedicated links,
- a partition class Padmin.

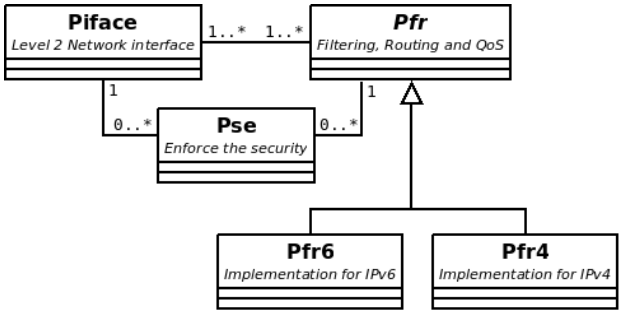

Figure 7 SNG Router partition classes

For demonstration purposes, the SNG Router will be in the following configuration: two network interfaces can only communicate in IPv4 and one of them needs some security features. The separation micro-kernel will thus run 5 partition instances: 2 instances of Piface, 1 of Pse, 1 of Pfr4 and 1 of Padmin. Note the Piface class is instantiated twice (one per network device) but will be designed only once. We do not need the Pfr6 partition class for this configuration. The object diagram in figure 8 sums up the different partition instances with their associated classes and their interconnections.

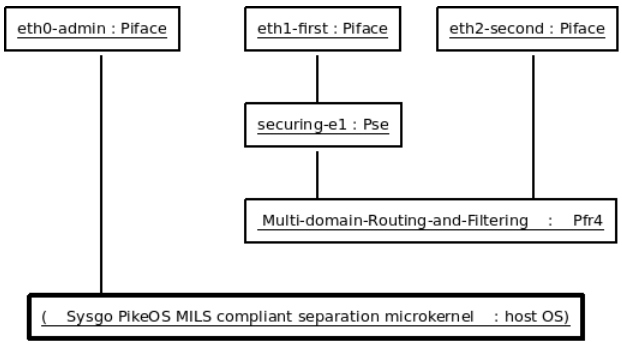

Figure 8 SNG Router partition instances

At the end of this step, the SNG Router development project is composed of a sub-project for each partition class and one sub-project to integrate

together the partitions into one standalone target. In the following step we focus around the Pfr4 partition class.

Design stage

In this part of the process, the designers have to implement the partition classes with Simulink and Stateflow models. Figure 9 shows partially the Pfr4 implemented in Simulink with different Stateflow chart sub-models and calling an external C function. Two implementations exist for this function: the first one writes the outputs to a file, enabling the designer to verify and debug the triggered outputs of Simulink; the second one is generated by the glue maker, connects the outputs of the GeneAuto library code with the outputs of the PikeOS partition and is thus an essential part of the final software.

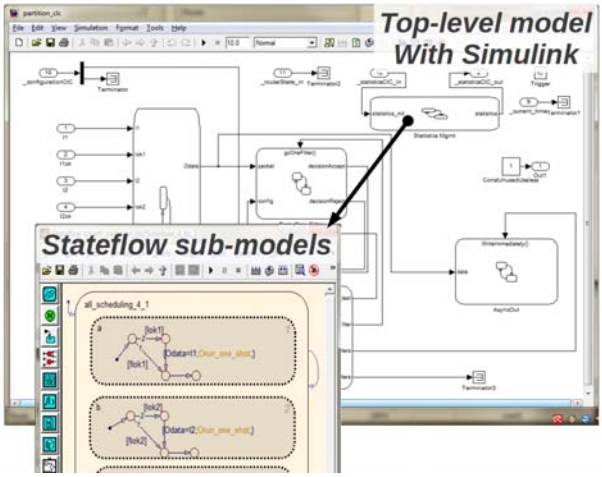

Figure 9 Modeling of the Pfr4 partition class

Transforming stage

After designing all the models for the different classes of partition, GeneAuto converts each model into C language code, resulting in a few minutes in thousand lines of codes in .c and .h files. For instance, figure 10 presents the conversion of a Stateflow sub-model.

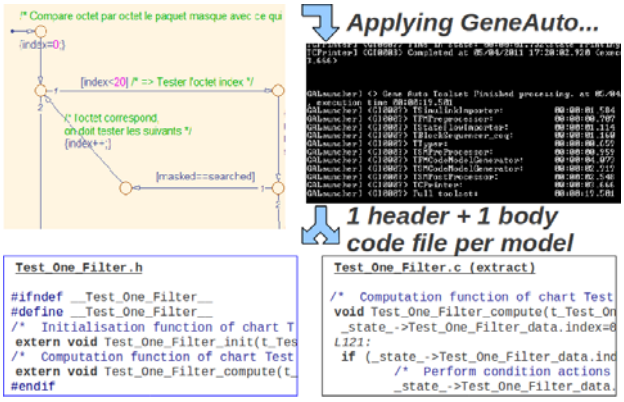

Figure 10 Generating the library code for Pfr4

The generated code is readable by a human but is not really interesting when you have the source graphical models.

Glueing stage

The generated code is some “library code”, that means there is no a single entry point a compiler requires for building applications. Then, we must apply the glue maker to generate some C lines of code in order to create the partition’s application, as illustrated in figure 11: it calls the function written by GeneAuto with the inputs and outputs of the partition class (this is the glue code regular usage illustrated in figure 5). In addition, the inputs reader and the outputs writer functions can be called at any time from the model: this is the optional triggered usage.

```
/* AUTOGENERATED GLUE CODE FOR PARTITION PFR4 */
/* Do not modify, modifications may be lost...*/
/* GLUE MAKER INPUTS: Pfr4_on_PikeOS.xml      */

/*****  PikeOS Specific part *****/
#include <vm.h>
#ifdef PIKEOSDEBUG
#include <vm_debug.h>
#endif
VM_DECLARE_STACK(0x4000)

/*****  Model specific part *****/
#include "pfr4.h"

/*****  Glue code part for Pfr4_on_PikeOS *****/
#include "pfr4_glue_io.h"
#define FOREVER while(1)

/* All partition inputs and outputs */
t_pfr4_io io;
/* State of this partition  */
t_pfr4_state st;

extern void _p4_entry(void) {
  vm_init();          /* Init PikeOS PSSW */
  pfr4_init(&st);     /* Init Pfr4 model  */

  FOREVER {           /*  <Regular usage> */
    read_all_pfr4_partition_inputs(&io);
    pfr4_compute(&io,&st);
    write_all_pfr4_partition_outputs(&io);
  }
}
```

Figure 11 Generating the glue code for Pfr4

Compiling stage

After having generated the source code for the partition classes, the next step consists of compiling it and generating the associated binary executable file. Then, this binary file shall be inserted and linked with the integration PikeOS project.

The CODEO plug-in eases this step: the developer creates one PikeOS-development project per partition, and then he includes the source files and completes some information such as the target architecture and processor and eventually the integration project. Then, PikeOS helper software generates an appropriate Makefile to compile the partition files.

The design, transforming, glueing and compiling steps must be done for each partition class before concluding the next integrating step. Figure 12 sums up these steps.

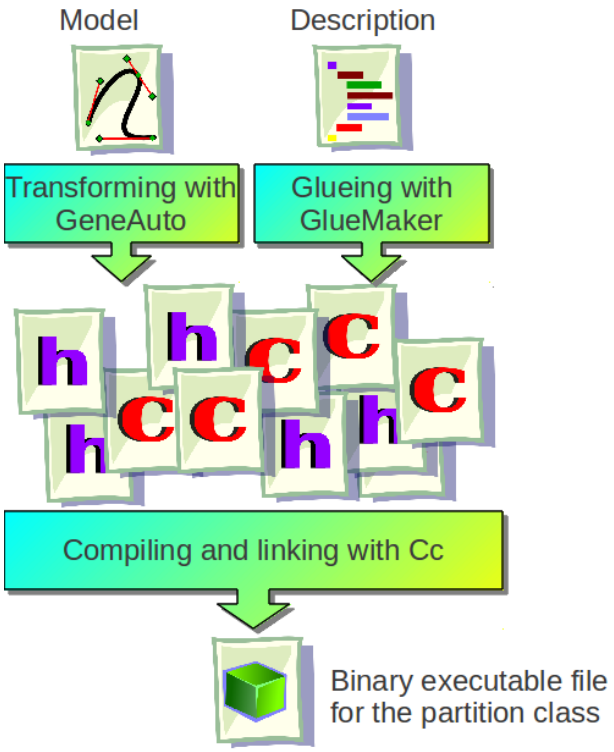

Figure 12 Transforming, glueing and compiling steps

Integrating stage

At this part of the process, we have the binaries for each partition class. So we quit the Pfr4 focus to refocus on the global view of the SNG Router. The integration project described in figure 13 needs to know where are the binaries for each class of partition, how many instances of each class the target will run, how they are scheduled, how they are prioritized... In addition, the architect must specify all communications allowed between the different instances, all shared memories and their access control lists, and how memory space is assigned to each instance… In fact, this PikeOS configuration step must be done by the architect after the first partitioning step and before the running step, it is not mandatory to have the partition binaries to complete the PikeOS integration project configuration. The CODEO plug-in is very helpful for the programmer for this task, because it writes directly the XML description file which may be a very long task if done manually.

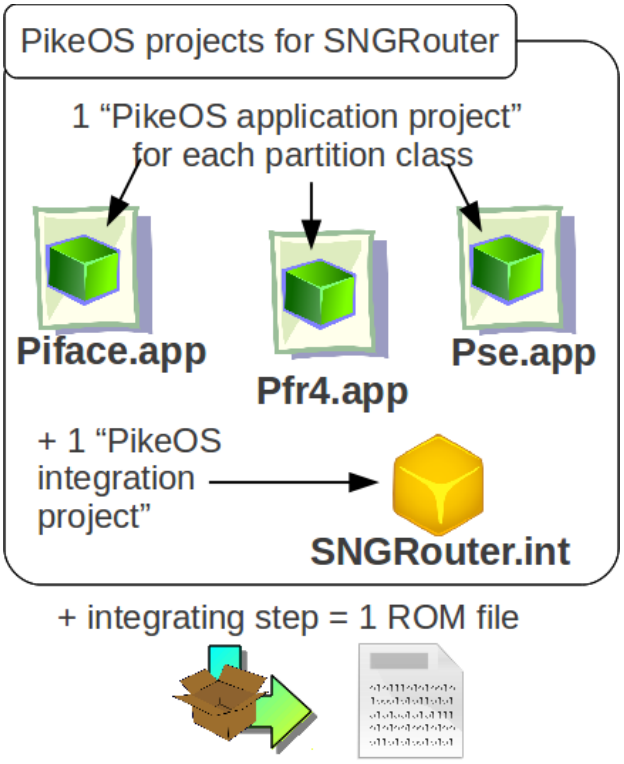

Figure 13 SNG Router integration with CODEO

After completing the PikeOS integration project configuration and inserting all the binaries, the final binary ROM image can be generated.

Running stage

Now the final binary file can be downloaded onto the target and run for additional tests or production usage. An emulator such as qemu can be used to check the software product or to instrument its execution: for instance with the help of the “Trace tool for PikeOS” and the CODEO monitor, the developer can control completely the execution of all parts of the Router, stop or restart some partitions. The snapshot of the Figure 14 illustrates this scenario.

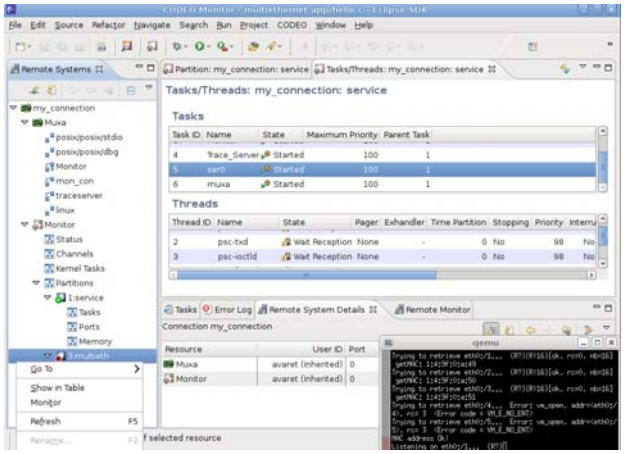

Figure 14 Snapshot of the final binary execution

Conclusion and future perspectives

Safe and secure software product development with high levels of assurance (DAL-A, EAL5) can be improved by the methodology we presented in this paper. The continuous enhancements in computer science make it possible now to unify model driven development techniques with virtualization infrastructures. It enables us to reduce development, certification and evaluation processes and simultaneously improve the safety and the security levels of the final product. The IP based Secure Next Generation Router case study we are conducting supports the methodology adjustments and provides a concrete application to extend the aeronautical network capabilities, such as data multiplexing through a single network link. The Open Source world provides some powerful tools such as GeneAuto, with the benefits we discussed previously for the industry.

To complete our SNG Router case study, we are working on network security requirements. A dedicated partition named Pse is already included in the design of our router and can be supplied with different functionalities we are elaborating.

However we have to conduct further research to extend the methodology with new tool chains. For instance, the formal research community provides some development tools for the B-Method. A tool chain “B” based on these tools should help the user with advanced safety and security assurance levels. In addition, DO-178C should soon clarify officially the certification of software with formal tools.

Another point we are working on concerns the overlap between evaluation and certification processes: the safety and the security assurances are conducted concurrently, based on independent standards. As a consequence some quite similar tasks must be conducted twice although the objectives and the methods are almost the same. Indeed, in the near future, we plan to propose some improvements to our methodology to reduce this overlap.

References

[1] RTCA SC-167, EUROCAE WG-12, 1992, DO-178B Software Considerations in Airborne Systems and Equipment Certification, Washington, DC, RTCA, Inc.

[2] July 2009, Common Criteria for Information Technology Security Evaluation, version 3.1, ISO/IEC 15408, http://www.niap-ccevs.org/cc-scheme/cc_docs/

[3] Huyck, Patrick, 2010, SKPP Conformance - Activities and Considerations to Achieve Certification, Salt Lake City, UT, 29th Digital Avionics Systems Conference, 4.A.6.1-8

[4] January 2007, ARINC 653 Avionics Application Standard Software Interface, Annapolis, MA, Aeronautical Radio, Incorporated

[5] Simulink, The MathWorks Inc., http://www.mathworks.com/products/simulink/

[6] Scicos, Metalau team of INRIA, http://www.scicos.org/

[7] Stateflow, The MathWorks Inc., http://www.mathworks.com/products/stateflow

[8] Safety Critical Application Development Environment (SCADE), Esterel Technologies, http://www.esterel-technologies.com/products/scade-suite/

[9] Toom, A., et al., 2008, Gene-Auto: an Automatic Code Generator for a safe subset of Simulink/Stateflow and Scicos, 4th European Congress ERTS Embedded Real Time Software

[10] Rugina A., et al., 2008, Gene-Auto: Automatic Software Code Generation for Real-Time Embedded Systems, Data Systems in Aerospace (DASIA) 2008

[11] Toom, A., et al., 2010, Towards Reliable Code Generation with an Open Tool: Evolutions of the Gene-Auto toolset, 5th European Congress ERTS Embedded Real Time Software

[12] Gaska, Thomas, and al., October 2010, Applying virtualization to avionics systems – The integration challenges, Salt Lake City, UT, 29th Digital Avionics Systems Conference, 5.E.1.1-8

[13] PikeOS embedded Virtualization, SYSGO AG, Germany, http://www.sysgo.com/products/pikeos-rtos-and-virtualization/embedded-virtualization/

[14] The WindRiver website, http://www.windriver.com/products/vxworks/

[15] Sysgo PikeOS, SYSGO AG, http://www.sysgo.com/products/pikeos-rtos-and-virtualization-concept/

[16] Atelier B website, http://www.atelierb.eu/

[17] The B0 Implementation Translation into C language ComenC website, ClearSy System Engineering, France, http://www.comenc.eu/

[18] The Caml language website, http://caml.inria.fr/index.en.html

[19] Oracle VM VirtualBox, an x86 virtualization software package, http://www.virtualbox.org/

[20] QEMU, a generic and open source machine emulator and virtualizer, http://wiki.qemu.org/

[21] CODEO, Eclipse based IDE, SYSGO AG., http://www.sysgo.com/products/pikeos-rtos-and-virtualization/eclipse-based-codeo/

Acknowledgements

We would like to thank Rupert Salmon and John Kennedy for their help in editing this paper. We express also our gratitude to the Gene-Auto team, especially to Marc Pantel, for the answers on the Gene-Auto forum.

Email Addresses

For further information, Antoine Varet can be contacted at antoine.varet@recherche.enac.fr and Nicolas Larrieu at nicolas.larrieu@enac.fr.